- способствуют быстрейшему профессиональному и
- духовному росту читателей и являются рекламой
- бумажных изданий таких документов.
-
- Все авторские права сохраняются за правообладателем.
- Если Вы являетесь автором данного документа и хотите
- дополнить его или изменить, уточнить реквизиты автора
- или опубликовать другие документы, пожалуйста,
- свяжитесь с нами по e-mail - мы будем рады услышать ваши
-

 пожелания.

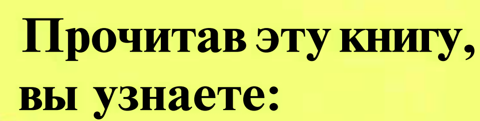
Вадим Дунаев

САМОУЧИТЕЛЬ JavaScript

Изучите один из популярнейших языков для веб-программирования самостоятельно

2-е издание





- общие принципы программирования
- основные элементы языка JavaScript
- алгоритмы и методы веб-программирования
- порядок разработки собственных приложений



Вадим Дунаев

(САМОУЧИТЕЛЬ)

JavaScript

2-е издание



Москва • Санкт-Петербург • Нижний Новгород • Воронеж
Новосибирск • Ростов-на-Дону • Екатеринбург • Самара
Киев • Харьков • Минск
2005

ББК 32.988.02-018я7
УДК 004.738.5(075)
Д83

Дунаев В.
Д83 Самоучитель JavaScript, 2-е изд. — СПб.: Питер, 2005. — 395 с.: ил.
ISBN 5-469-00804-5

Эта книга предназначена для самостоятельного освоения программирования на языке JavaScript. Кроме общего руководства, она содержит множество примеров и текстов готовых к использованию программ. Рассматриваются вопросы создания сценариев для веб-сайтов, атакже сценариев, выполняемых Windows Scripting Host. В приложениях приводится справочная информация по JavaScript и HTML. Книга адресована как новичкам, так и тем, кто уже имеет некоторый опыт в веб-дизайне и программировании. Во втором издании книги исправлены замеченныеопечаткиинеточности.

ББК 32.988.02-018я7
УДК 004.738.5(075)

ISBN 5-469-00804-5

Краткое содержание

Содержание

От автора

После выхода в свет моих книг «Сам себе web-мастер» (2001) и «Сам себе web-дизайнер» (2002) многочисленные читатели часто спрашивали меня о том, как написать тот или иной сценарий на языке JavaScript для поддержки своей веб-страницы. Сначала я пытался оперативно отвечать на все вопросы, но вскоре вынужден был оставить это занятие — просто не успевал. Вместе с тем появлением этой книги я обязан вам, уважаемые читатели. Понятно, что, освоив HTML, начинающие веб-дизайнеры сначала ищут по всему свету образцы недостающих им сценариев, а затем — рецепты их создания. Рано или поздно они ощущают необходимость глубже познать язык сценариев, чтобы не гоняться за рецептами, а самим их создавать. Кроме того, на разработку веб-сайтов можно взглянуть не только с традиционной точки зрения HTML, но и с позиций языка сценариев, что еще более увлекательно. Как и HTML, язык JavaScript с успехом претендует на роль народного языка общения с компьютером. Он очень популярен, его легко изучать не искушенным в программировании новичкам. Тем не менее он достаточно выразителен при создании вполне серьезных и полезных приложений как для Веб, так и для локальных компьютеров.

Хотя JavaScript прост для усвоения и практического использования, полное его описание занимает более 1000 страниц. Я понимаю, что далеко не каждый, даже весьма заинтересованный этой темой, отважится на чтение такого фолианта, особенно если ему предстоит все начинать с нуля. Эта книга — не справочник, а практическое руководство для самостоятельного изучения JavaScript, не требующее от читателя предварительной подготовки в области программирования. Я хотел сделать ее небольшой, понятной и полезной. В ней содержится масса сведений, рекомендаций и даже рецептов, которые вам могут потребоваться на первых порах создания веб-сайтов и локальных приложений. Однако главная цель книги — помочь научиться программировать. Если вы никогда ранее не занимались программированием, то эта книга даст вам ключ к освоению не только JavaScript, но и любого другого языка программирования. В частности, от JavaScript вы сможете довольно легко перейти к изучению С++, Java и др. Вообще говоря, стоит только освоить один язык, и вам открыты двери к любому другому. Если вы начинаете с JavaScript, то это очень хороший выбор.

На каком-то этапе проникновения в мир программирования на JavaScript вам, вероятно, потребуются дополнительные сведения. Осилив предлагаемую книгу, вы уже будете готовы к их восприятию. Вот тут-то вам и понадобятся справочники с детальной и полной информацией, которые найдутся в книжных магазинах,

на полках библиотек или в Интернете. В настоящее время наилучшим, на мой взгляд, справочником по JavaScript является весьма объемная книга Д. Гудмана «JavaScript. Библия для пользователя».

Материал предлагаемой книги поддерживается моим сайтом, расположенным по адресу www.admiral.ru/~dunaev. По мере сил и возможностей я периодически обновляю его. Свои отзывы, вопросы и пожелания вы можете оставить в гостевой книге. Так что добро пожаловать!

Благодарность

Я искренне благодарю свою жену Валентину за понимание и внимательное отношение ко мне в период работы над книгой.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу электронной почты comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства http://www.piter.com вы найдете подробную информацию о наших книгах.

Введение

Язык программирования, как и любой другой язык (естественный, например), предназначен для коммуникации, то есть связи между говорящим и слушающим. В программировании говорящим является программист, а слушателем — интерпретатор языка, некоторая компьютерная программа, понимающая этотязык и выполняющая действия в соответствии с тем, что она поняла.

Было время, когда считалось, что для облегчения общения с компьютером необходимо создать язык, достаточно близкий к естественному. Эта идея в конечном счете не выдержала испытаний временем, хотя и породила несколько прекрасных языков программирования. Часто бывает, что побочные эффекты некоей деятельности превосходят ожидания. Мы испытываем трудности общения и на естественном языке; что уж говорить о специализированных языках, тем более о языках общения с компьютером... Проблема решается на удивление просто. Главное здесь — понять корень или причину проблемы. Вся сложность заключается не в языке, а в культуре или способе его употребления. В конце концов, язык представляет собой лишь нормативную базу, а люди используют ее лишь так и в той мере, как смогли или как им показалось достаточным для своих житейских нужд. Когда люди, сами того не желая, размывают небрежной речью нормативную базу языка, они разрушают систему коммуникации не только между собой, но и между последующими поколениями. А как соблазнительно использовать слэнг и жаргон! При этом возникает иллюзия причастности к особой, исключительной группе, некоей профессиональной касте. Кажется, что это выделяет нас из толпы. Понятно, что для становления личности необходимо выделиться из среды, но не настолько, чтобы выпасть на обочину магистрали, по которой движется все наше общество. На обочине вас никто не заметит, пролетая мимо, а вам самим останется только копошиться в грязи, чтобы потом рассказать кому-нибудь о перипетиях вашего лихого путешествия по жизни. Но соблазны на то нам и даны, чтобы мы могли показать свою способность им противостоять.

Любой язык, на котором хотя бы кто-нибудь говорит, — живой. Практика использования языка стимулирует развитие его нормативной базы и тем самым поддерживает его существование. Владимир Даль не описывал нормы русского языка, а фиксировал словоупотребление, поэтому его словарь и называется словарем «живого великорусского языка». Что касается искусственных языков, то есть языков программирования, то трудности их освоения новичками обусловлены, главным образом, недостатком практики программирования. Стандартный (формальный) способ описания языка хорош для тех, кто уже изучал любой язык программиро-

вания. Для новичков же стандартный, тем более формальный способ изложения правил языка является, мягко говоря, неприемлемым. Мы постигаем нечто иначе, чем излагаем уже освоенное. Эта хорошо известная истина, к сожалению, почему-то часто забывается и учениками, и преподавателями. Однако авторы почти всех руководств по языку, снискавших широкое признание и добившихся успеха, следовали этому основному принципу.

Давайте для пробы пера напишем и выполним хотя бы элементарную, но все же полноценную программу на JavaScript, чтобы убедиться в реальности нашей затеи изучить программирование. Откроем простой текстовый редактор, например Блокнот (Notepad) Windows (рис. B.I), и введем с клавиатуры следующий текст:

```
WScript.echo("Привет!")
```

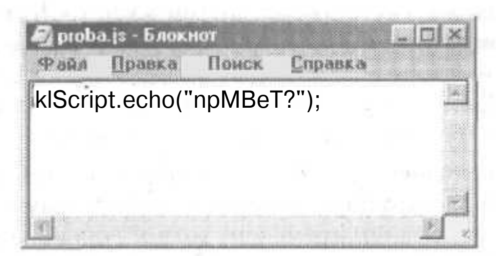


Рис. B.I. Код программы в окне текстового редактора

Мы написали элементарную программу на языке JavaScript, состоящую из одной строки. Сохраним эту программу в файле под именем, например, proba.js. Имя файла может быть произвольным, а вот расширение js указывает, что он содержит выражения, написанные на языке JavaScript. Чтобы сохранить файл, выполните команду меню Файл ▸ Сохранить как. В открывшемся диалоговом окне из раскрывающегося списка Тип файла выберите Все файлы (*.*), а в поле Имя файла введите с клавиатуры имя файла с расширением js, например proba.js (рис. B.2). В заключение щелкните на кнопке Сохранить.

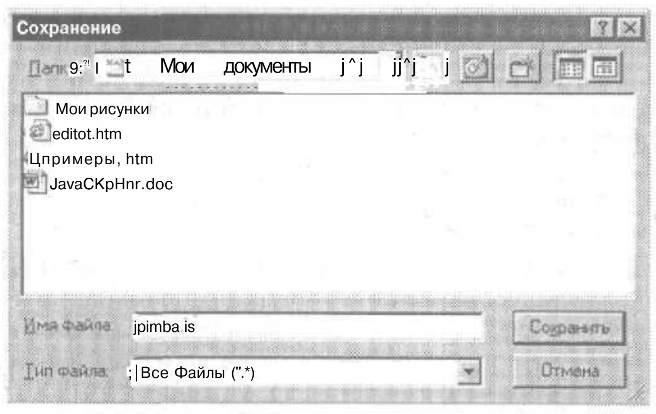


Рис. B.2. Диалоговое окно сохранения файла

Итак, наша программа написана и сохранена в файле. Чтобы ее выполнить, в Проводнике Windows дважды щелкните на имени файла proba.js. В результате на экране появится диалоговое окно со словом «Привет!» и кнопкой OK (рис. B.3). Щелчок на этой кнопке закроет окно. Так работает наша программа.

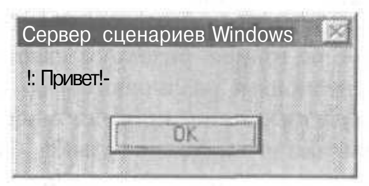


Рис. B.3. Диалоговое окно с сообщением, созданное программой

Теперь немного модифицируем нашу программу. Откроем Блокнот и напишем в нем следующие строки:

```
<html>
<script>
alert("Привет")
</script>
</html>
```

Сохраним эту программу в файле с именем proba.htm. Если мы ее выполним, то в результате откроется окно веб-браузера, а на его фоне — диалоговое окно со словом «Привет!» и кнопкой OK (рис. B.4).

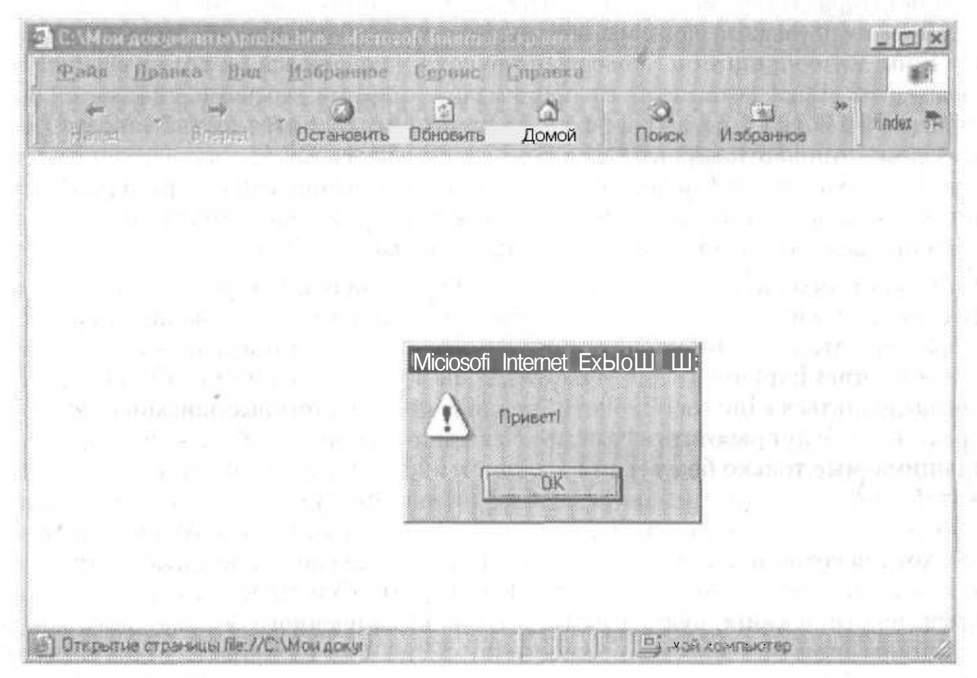


Рис. B.4. Диалоговое окно с сообщением на фоне браузера Internet Explorer

Итак, эффекты выполнения рассмотренных выше двух программ почти одинаковы: обе программы выводят на экран диалоговое окно с сообщением. Различие состоит в том, что в первом случае диалоговое окно с сообщением появляется само по себе, а во втором — на фоне окна браузера. Дело в том, что эти почти одинаковые программы исполняются различными интерпретаторами языка. В первом случае это сервер сценариев Windows, называемый Windows Scripting Host, а во втором — веб-браузер. Вообще говоря, редакции языка JavaScript, понимаемые этими интерпретаторами, несколько различаются, но между ними много общего — в смысле как синтаксиса, так и принципов построения. Поэтому можно говорить о различных редакциях одного и того же языка.

Таким образом, мы столкнулись с тем, что различные исполнительные системы (интерпретаторы) воспринимают различные редакции одного и того же языка. Это происходит из-за того, что только благодаря интерпретаторам можно говорить о языке как о чем-то реальном. Представьте себе следующую ситуацию. Допустим, мы сами придумали некий язык программирования. Чтобы на нем действительно можно было писать программы, необходимо создать исполнительную систему — интерпретатор этого языка. Интерпретатор сам является программой, написанной на другом языке программирования и, возможно, другими людьми. Он может воспринимать декларированный нами язык в полной мере или только частично. А может случиться и так, что некоторые элементы языка интерпретатор будет воспринимать по-своему, не так, как это изначально предполагалось авторами. Поэтому появляются различные редакции языка, каждая для своего интерпретатора. Например, редакции JavaScript, понимаемые Microsoft Internet Explorer и браузерами Netscape, значительно отличаются друг от друга. С другой стороны, интерпретаторы усовершенствуются, появляются их новые версии. Те фрагменты языка, которые не воспринимались старыми версиями, становятся доступными в новых. Иначе говоря, интерпретаторы стремятся соответствовать некоторому стандартному подмножеству языка. Язык также может развиваться с учетом практики его использования и проблем, возникающих при разработке интерпретаторов. Постепенно формируется стандарт языка, который должен поддерживаться интерпретаторами различных производителей.

На момент выхода в свет этой книги современной являлась версия 1.5 JavaScript. Все рассмотренные в книге конструкции языка и примеры сценариев работают в браузере Microsoft Internet Explorer 6.0 для Windows; многие из них подходят и для Internet Explorer 4+ для Windows, но не все, — возможно, они будут корректно выполняться в Internet Explorer 5 для Macintosh. Некоторые описанные в книге средства не будут работать в Netscape Navigator версии 4 и старше. Средства, воспринимаемые только браузерами, отличными от Internet Explorer для Windows, в этой книге не рассматриваются вообще. Если не оговаривается особо, то описываемые в книге средства применимы для Internet Explorer 4 для Windows и старше, хотя некоторые из них могут работать и в более ранних версиях. Если у вас установлен Internet Explorer для Windows версии 5.5 и старше, то все сценарии, приведенные в книге, будут работать. Из-за ограниченного объема книги в ней отражены лишь основные возможности JavaScript, кое-что пришлось оставить без внимания. Если вы чего-то не найдете, то обращайтесь к другим, более специализированным или более объемным книгам.

Глава 1 посвящена основным элементам собственно языка JavaScript. Глава 2 знакомит читателя с основными понятиями и важнейшими объектами, используемыми при создании сценариев, которые выполняются браузером. По существу, это введение в мир разработки сценариев. Обзор объектной модели браузера и документа приводится в главе 3. Материал этой главы можно использовать в качестве краткого справочника и пропустить при первом чтении. Он понадобится для получения ориентации в пространстве объектов. Примеры сценариев, решающих конкретные практические задачи, представлены в главе 4. Хотя примеры присутствуют во всех главах, в этой главе они иллюстрируют комплексное применение различных средств JavaScript и приближены к типовым задачам веб-дизайна. Глава 5 посвящена работе с файловой системой компьютера и реестром Windows с помощью программ на JavaScript. Ее можно читать отдельно, поскольку материал не связан с предыдущими главами. В приложении приведены руководство и справочник по HTML.

Изучение материала книги может быть успешным только при активном отношении читателя. Хотя описываемые в книге программы вполне работоспособны, их коды, скорее всего, несовершенны и могут быть улучшены, модифицированы для решения других задач либо интегрированы с другими программами в каком-то сложном программном проекте. Если вы попробуете что-либо с ними сделать, то многому научитесь и многое поймете. Это — магистральный путь познания программирования. Итак, мы начинаем.

Глава 1. Основы JavaScript

1.1. Где писать программы и как их запускать

В этой главе мы рассмотрим основы языка JavaScript: важнейшие понятия и синтаксис (правила построения выражений). Приводимые примеры желательно самостоятельно выполнить на компьютере. Это можно сделать по-разному, но по крайней мере на первом этапе я рекомендую воспользоваться самым простым и доступным способом: в качестве интерпретатора (исполнительной системы) программ на JavaScript возьмите веб-браузер. Я работал с Internet Explorer 6.0. Большинство приведенных в книге примеров работают и в версии 5.0 этого браузера. В качестве редактора ваших программ выберите какой-нибудь простой текстовый редактор, например Блокнот Windows. Откройте текстовый редактор и создайте в нем заготовку файла, который вы будете потом редактировать, вводя экспериментальные выражения или даже целые программы. А именно введите в рабочее поле редактора следующий текст:

```
<HTML>
<HEAD><TITLE>npMMepbi</TITLE></HEAD>
<SCRIPT>

</SCRIPT>
</HTML>
```

По существу, это HTML-документ, определяющий пустую веб-страницу. Сохраните этот файл на диске, например под именем пример.htm. Расширение имени файла должно быть htm или html, поскольку мы хотим, чтобы он исполнялся в веб-браузере. В этом файле мы написали теги языка HTML. Выражения на языке JavaScript следует записывать между тегами <SCRIPT> и </SCRIPT>. При изучении этой главы записывайте в каждой строке не более одного выражения на языке JavaScript. Заканчивайте строку нажатием на клавишу Enter, чтобы перейти к новой строке. Ниже приведен пример программы:

```
<HTML>
<HEAO><T1T1E>Примеры< /TITLE ></HEAD>
<SCRIPT>
x = 5
y = x + 3
alert(y)

</SCRIPT>
</HTML>
```

При выполнении учебных примеров обычно требуется вывести на экран окончательные или промежуточные данные. Для этого можно использовать метод alertQ, указав в круглых скобках то, что требуется вывести на экран. В приведенном выше примере метод alert(y) выведет на экран диалоговое окно, в котором отобразит значение переменной y (в данном случае — число 8). Если не вставлять строку alert(y), то программа выполняется без отображения результата. Если вы хотите выводить результаты вычислений не в диалоговом окне, а непосредственно в окне веб-браузера, то вместо alert(y) напишите следующее выражение:

```
document.writeln(y)
```

Чтобы открыть файл (в нашем случае это примеры.htm) в веб-браузере и выполнить содержащуюся в нем программу, достаточно просто дважды щелкнуть левой кнопкой мыши на нем в окне Проводника Windows. Если теперь потребуется открыть этот файл в окне текстового редактора, то щелкните правой кнопкой мыши где-нибудь на свободном пространстве в окне браузера и в контекстном меню выберите команду **Просмотр в виде HTML** Можно также воспользоваться и командой меню **Вид > В виде HTML** В результате откроется окно текстового редактора с текстом вашей программы (рис. 1.1). Внесите необходимые изменения и сохраните их на диске (Файл > Сохранить). Чтобывеб-браузерсмоготобразитьизмененный файл, выполните команду Вид > Обновить (рис. 1.2). Таким образом, переходя от текстового редактора к браузеру и обратно, можно разрабатывать и отлаживать свои программы.

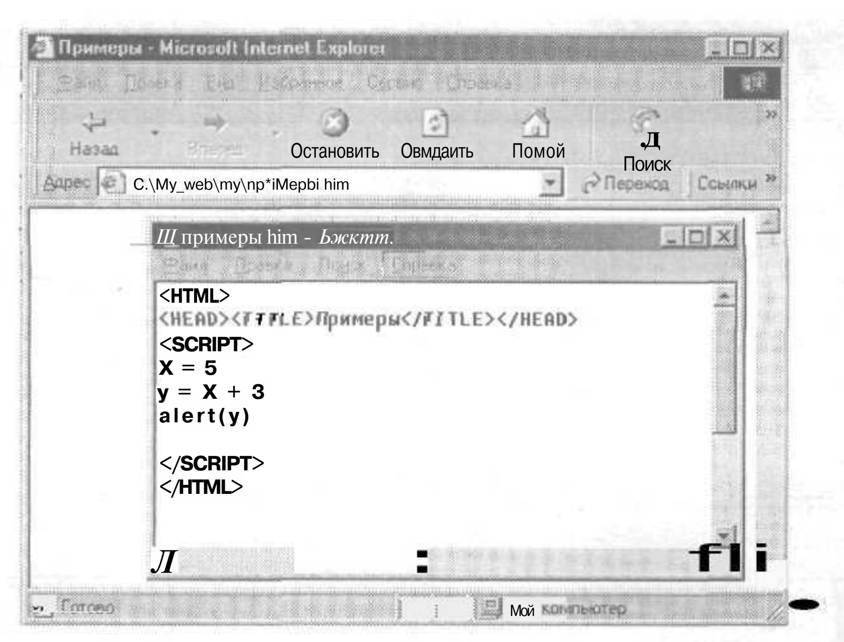


Рис. 1.1. Текстовый редактор Блокнот с программой на фоне браузера Internet Explorer

Создавать программы на JavaScript можно и с помощью программ, специально предназначенных для разработки веб-сайтов, — например Microsoft FrontPage или Macromedia Dreamweaver (рис. 1.3). Однако здесь мы не будем рассматривать эти средства.

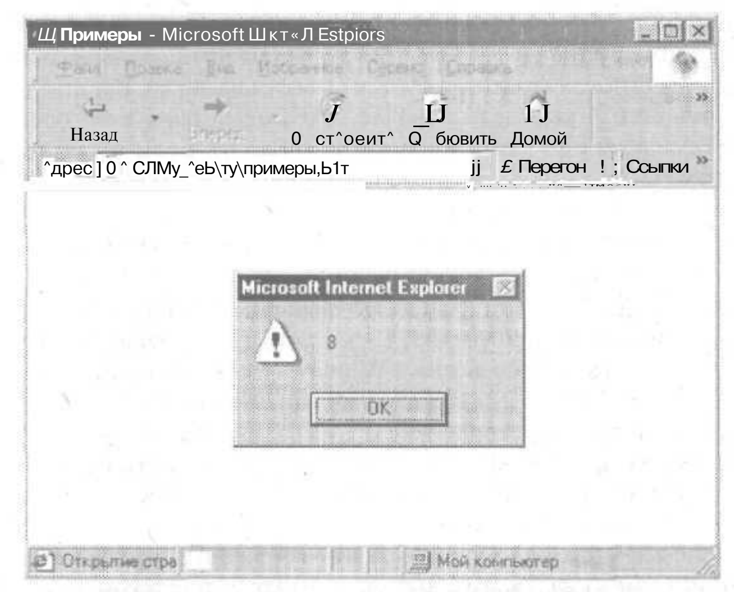


Рис. 1.2. Результат выполнения программы из файла пример.htm

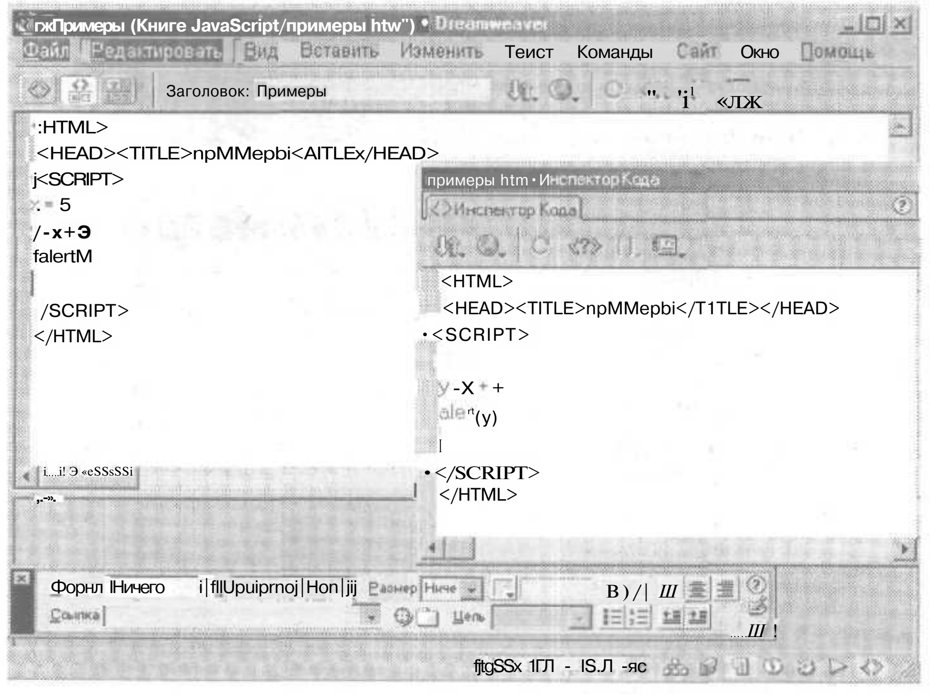


Рис. 1.3. Окно Dreamweaver 4.0 — программы для разработки веб-сайтов

Наконец, я предлагаю вам быстро создать свой собственный редактор программ. Для этого следует написать небольшой HTML-код и сохранить его на диске

в файле с расширением .htm или .html. Этот файл можно открыть в браузере как обычную веб-страницу. Вот код предлагаемого редактора:

```
<HTML>
<H3>Редактор кодов JavaScript</H3>
Код:<Ы>
<TEXTAREA id="mycode" ROWS=10 COLS=60></TEXTAREA>
<p>Результат:<Ы>
<TEXTAREA id="myrezult" ROWS=3 COLS=60></TEXTAREA>
<P>
<BUTTON onclick="document.all.myrezult.value=eval(mycode.value)">
Выполнитb</BUTTON>
<BUTTON onclick="document.all.mycode.value='';
document.all.myrezult.value=''">
04HCTHTb</BUTTON>
<P>
<! Комментарий >
Введите выражения в верхнее поле.
Выражения разделяются точкой с запятой.
Можно также писать каждое выражение в отдельной строке.
Чтобы закончить одну строку и перейти к другой, нажмите клавишу Enter.
</HTML>
```

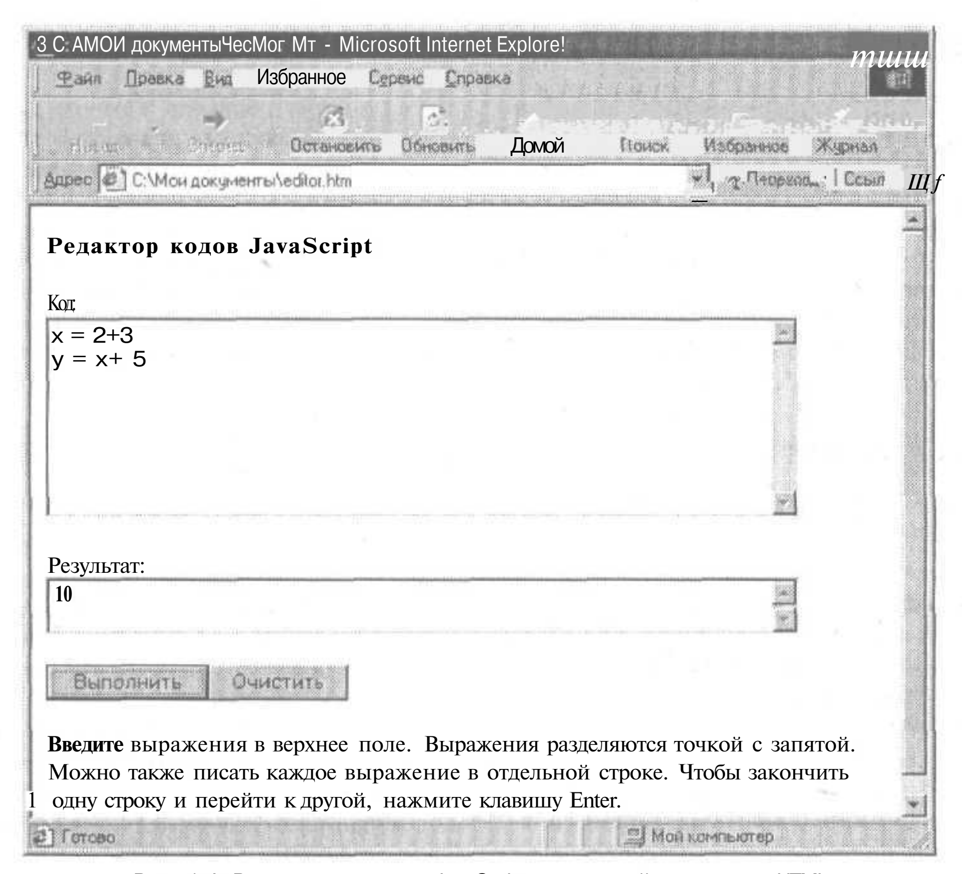


Рис. 1.4. Редактор программ JavaScript, сделанный с помощью HTML

В веб-браузере этот HTML-код выведет две текстовых области: одна служит для • ввода выражений на языке JavaScript, а другая — для отображения результатов этого кода. Кроме текстовых областей создаются две кнопки: одна для выполнения введенного кода и отображения результатов, а вторая — для очистки текстовых областей. В поле результатов выводится значение, возвращаемое последним выражением введенного кода. Обратите внимание, что в поле для кода следует вводить только выражения на языке JavaScript, но не теги HTML.

Здесь мы не будем разбирать, как устроен предлагаемый код редактора. Отметим лишь, что в его основе лежат простые сценарии на языке JavaScript, выполняющие роль обработчиков события, например щелчка кнопкой мыши (onclick). На рис. 1.4 показано, как выглядит наш редактор в окне браузера Internet Explorer.

СОВЕТ

Если вы решитесь использовать самодельный редактор, то при возникновении ошибок в ваших программах (сообщение об ошибке отображается в статусной строке веб-браузера) выполните команду меню браузера Вид > Обновить.

1.2. Ввод и вывод данных

В JavaScript предусмотрены довольно скудные средства для ввода и вывода данных. Это вполне оправданно, поскольку JavaScript создавался в первую очередь как язык сценариев для веб-страниц. Основой веб-страниц является код, написанный на языке HTML, который специально рассчитан на форматирование информации и создание пользовательского интерфейса. Поскольку сценарии на JavaScript хорошо интегрируются с HTML-кодом, постольку для ввода и вывода данных вполне подойдут средства HTML Если вы пишете программу наJavaScript, которая будет выполняться веб-браузером Internet Explorer, то можете воспользоваться тремя стандартными методами для ввода и вывода данных: **alertQ, promptQ** и confirmQ. Рассмотрим эти методы браузера подробнее.

1.2.1. alert

Данный метод позволяет выводить диалоговое окно с заданным сообщением и кнопкой ОК. Синтаксис соответствующего выражения имеет следующий вид:

```
a1eГгТ("сообщение)
```

Если ваше сообщение конкретно, то есть представляет собой вполне определенный набор символов, то его необходимо заключить в двойные или одинарные кавычки. Например, **alert("Привет всем!")** (рис. 1.5). Вообще говоря, *сообщение* представляет собой данные любого типа: последовательность символов, заключенную в кавычки, число (в кавычках или без них), переменную или выражение.

Диалоговое окно, выведенное на экран методом **alert(),** можно убрать, щелкнув на кнопке ОК. До тех пор пока вы не сделаете этого, переход к ранее открытым окнам невозможен. Окна, обладающие свойством останавливать все последующие действия пользователя и программ, называются модальными. Таким образом, окно, создаваемое посредством alertQ, является модальным.



Рис. 1.5. Диалоговое окно, созданное методом a!er1("Привет всем!")

Выше мы уже говорили, что метод a1er±()можно использовать для вывода промежуточных и окончательных результатов программ при их отладке. При этом вы можете вывести результат вычисления какого-либо выражения и приостановить дальнейшее выполнение работы программы до тех пор, пока не щелкнете на кнопке OK.

1.2.2. confirm

Метод confirm позволяет вывести диалоговое окно с сообщением и двумя кнопками — OK и Отмена (Cancel). В отличие от метода alert этот метод возвращает логическую величину, значение которой зависит от того, на какой из двух кнопок щелкнул пользователь. Если он щелкнул на кнопке OK, то возвращается значение true (истина, да); еслиже он щелкнул на кнопке Отмена, то возвращается значение false (ложь, нет). Возвращаемое значение можно обработать в программе и, следовательно, создать эффект интерактивности, то есть диалогового взаимодействия программы с пользователем. Синтаксис применения метода confirm имеет следующий вид:

```
confirт(сообщение)
```

Если ваше сообщение конкретно, то есть представляет собой вполне определенный набор символов, то его необходимо заключить в кавычки, двойные или одинарные. Например, confirm("Bbi действительно хотите завершить работу?") (рис. 1.6). Вообще говоря, *сообщение* представляет собой данные любого типа: последовательность символов, заключенную в кавычки, число (в кавычках или без них), переменную или выражение. Если вы еще не знаете, что такое переменная и выражение, то вскоре восполните этот пробел.

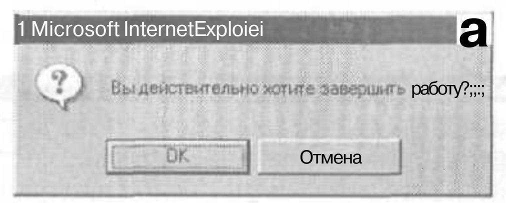


Рис. 1.6. Диалоговое окно, созданное методом confirm

Диалоговое окно, выведенное на экран методом confirm(), можно убрать щелчком на любой из двух кнопок — OK или Отмена. До тех пор пока вы не сделаете этого, переход к ранее открытым окнам невозможен. Окна, обладающие свойством останавливать все последующие действия пользователя и программ, называются модальными. Таким образом, окно, создаваемое посредством confirmQ, является

модальным. Если пользователь щелкнет на кнопке OK, то метод вернет логическое значение true (истина, да), а если он щелкнет на кнопке Отмена, то возвращается логическое значение **false** (ложь, нет). Возвращаемое значение можно затем обработать в программе и, следовательно, создать эффект интерактивности, то - есть диалогового взаимодействия программы с пользователем.

1.2.3. prompt

Метод prompt позволяет вывести на экран диалоговое окно с сообщением, а также с текстовым полем, в которое пользователь может ввести данные (рис. 1.7). Кроме того, в этом окне предусмотрены две кнопки: OK и Отмена **(Cancel). В** отличие от методов **alertQ** и **confirm()** данный метод принимает два параметра: сообщение и значение, которое должно появиться в текстовом поле ввода данных по умолчанию. Если пользователь щелкнет на кнопке O K, то метод вернет содержимое поля ввода данных, аесли он щелкнет на кнопке Отмена, то возвращается логическое значение false (ложь, нет). Возвращаемое значение можно затем обработать в программе и, следовательно, создать эффект интерактивности, то есть диалогового взаимодействия программы с пользователем. Синтаксис применения метода prompt имеет следующий вид:

prompt(сообщение, значение_поля_ввода_данных)

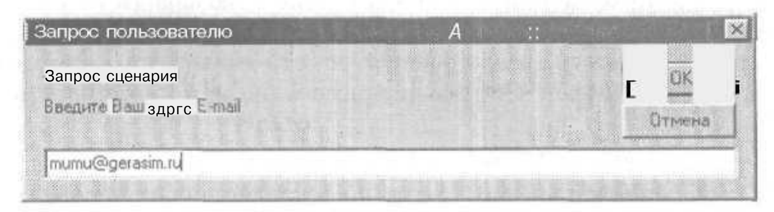


Рис. 1.7. Диалоговое окно, созданное методом prompt

Параметры метода prompt() не являются обязательными. Если вы их не укажете, то будет выведено окно без сообщения, а в поле ввода данных подставлено значение по умолчанию — undefined (не определено). Если вы не хотите, чтобы в поле ввода данных появлялось значение по умолчанию, то подставьте в качестве значения второго параметра пустую строку "". Например, prompt("Введите Ваше имя, **пожалуйста",** "")(рис. 1.8).

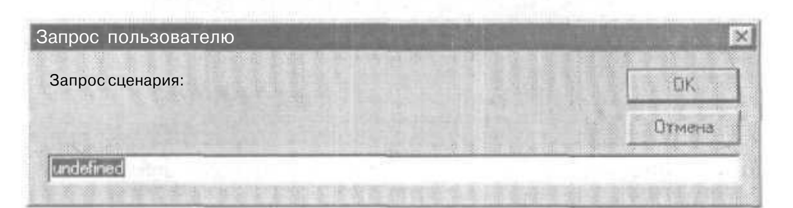


Рис. 1.8. Диалоговое окно, созданное методом promptQбез параметров

Диалоговое окно, выведенное на экран методом promptQ, можно убрать щелчком на любой из двух кнопок— OKили Отмена. Как и в случае использования метода **confirmQ,** переход к ранее открытым окнам невозможен.

1.3. Типы данных

В любом языке программирования очень важно понятие типа данных. Если не осознать его с самого начала, то потом придется часто сталкиваться со странным поведением созданной вами программы. Поэтому мы рекомендуем внимательно отнестись к данному разделу, чтобы потом не возвращаться к нему, теряя время на элементарные вещи. (Табл. 1.1.)

Данные, которые хранятся в памяти компьютера и подвергаются обработке, можно отнести к различным типам. Понятие типа данных возникает естественным образом, когда необходимо применить к ним операции обработки. Например, операция умножения применяется к числам, то есть к данным числового типа. Это известно еще из начальной школы. А что получится, если умножить слово «Вася» на число 5? Поскольку трудно дать вразумительный ответ на этот вопрос, напрашивается вывод: некоторые операции не следует применять к разнотипным данным. Мы также не знаем, что должно получиться в результате умножения слов «Вася» и «Маня», поэтому заключаем, что определенные операции вообще не применимы к данным некоторых типов. С другой стороны, существуют операции, результат которых зависит от типа данных. Например, операция сложения, обозначаемая символом «+», может применяться и к двум числам, и к двум строкам, состоящим из произвольных слов. В первом случае результатом применения этой операции будет некоторое число, а во втором — строка, получающаяся путем приписывания второй строки к концу первой. В случае строк операцию сложения еще называют склейкой или конкатенацией. Операции, применимые к различным типам данных, но обозначаемые одним и тем же символом, называют также перегруженными. Так, операция, обозначаемая символом «+», является перегруженной: применительно к числам она выполняет арифметическое сложение этих чисел, а применительно к строкам символов — склейку (приписывание, конкатенацию).

Таблица 1.1. Типы данных в JavaScript

Тип данных	Примеры	Описание значений
Строковый или символьный (string)	"Привет" "д.т. 123-4567"	Последовательность символов, заключенная в кавычки, двойные или одинарные
Числовой (number)	3.14 -567 +2.5	Число, последовательность цифр, перед которой может быть указан знак числа (+ или –); перед положительными числами не обязательно ставить знак «+»; целая и дробная части чисел разделяются точкой. Число записывается без кавычек
Логический (булевский, boolean)	true false	true (истина, да) или false (ложь, нет); возможны только два значения
Null		Отсутствие какого бы то ни было значения
Объект (object)		Программный объект, определяемый своими свойствами. В частности, массив также является объектом
Функция (function)		Определение функции — программного кода, выполнение которого может возвращать некоторое значение

На данном этапе следует обратить особое внимание на различия в представлении числовых и строковых (символьных) данных. Числа как данные числового типа всегда представляются без кавычек. Для их написания мы используем цифры и, при необходимости, знак числа и разделительную точку. Строковые данные заключаются в кавычки, двойные «"» или одинарные «'». Например, запись -345.12 представляет число, а запись "-345.12" — строку символов. Это данные различных типов, хотя мы и можем сказать, что их содержанием является одно и то же число. Но это не что иное, как обыденная интерпретация данных или, как еще говорят, семантика (смысл) данных. С точки зрения языка программирования число (точнее, данные числового типа) можно корректно использовать в арифметических операциях, а строки — в строковых операциях. Что означает «корректно использовать»? То, что использование данных в языке программирования должно соответствовать нашим традициям, не связанным с программированием. Например, обращение с числами корректно, если оно подчиняется правилам математики; обращение со строками корректно, если не противоречит правилам редакторской правки текста (вставка, удаление, склейка фрагментов и т. п.). Язык программирования призван обеспечить в той или иной мере выполнение операций, имеющих аналоги в обычной человеческой деятельности. Этой цели служит, в частности, и понятие типов данных.

Заметим, что строка "", не содержащая ни одного символа (даже пробела), называется пустой. При этом строка, содержащая хотя бы один пробел (например," "), не пуста.

Данные логического типа могут иметь одно из двух значений: true или false. Эти значения записываются без кавычек. Значение true означает истину (да), a false - ложь (нет). Обычно эти значения получаются в результате вычисления выражений с использованием операций сравнения двух данных, а также логических операций (И, ИЛИ, НЕ). Например, результатом вычисления выражения 2 < 3 является, очевидно, значение true (действительно, число 2 меньше числа 3). Логический тип данных называют еще булевским (boolean) в честь английского математика Джона Буля, придумавшего алгебру для логических величин.

Другие типы данных (Null, Object и Function) будут рассмотрены в следующих разделах, после изучения основ языка.

При создании программ на JavaScript за типами данных следит сам программист. Если он перепутает типы, то интерпретатор не зафиксирует ошибки, а попытается привести данные к некоторму типу, чтобы выполнить указанную операцию. Вам следует разобраться, к какому именно типу приводится смесь данных различного типа. Многие языки программирования, в том числе С и Pascal, не обладают этим свойством, они требуют явного указания типа данных.

Например, если вы напишете выражение 5+"Вася", то результатом его вычисления будет строка символов "5Вася". Таким образом, интерпретатор, столкнувшись с выражением сложения числа и строки символов, переводит число в строку, содержащую это число, а затем выполняет операцию сложения двух строк. Сложение двух строк в JavaScript дает в результате строку, получающуюся путем приписывания второй строки к концу первой.

Результатом вычисления выражения 2+3 будет число 5, а выражения **2+"3"** — строка "23", состоящая из двух цифровых символов. Как нетрудно заметить, в случае применения операции сложения к числу и строке символов интерпретатор преобразует число в строку символов и возвращает результат вычисления выражения тоже в виде строки символов. Иначе говоря, в случае смысловой несогласованности типов данных интерпретатор использует некоторые правила их согласования, принятые по умолчанию. В результате могут появиться трудно выявляемые ошибки. С другой стороны, эту особенность языка можно использовать для написания изящных и компактных кодов, соблюдая известную осторожность. Лично мне нравятся языки с так называемыми свободными типами данных.

Для преобразования строк в числа в JavaScript предусмотрены встроенные функции parselntQ и parseFloatQ. Что такое функция, мы подробно расскажем ниже в одном из разделов этой главы. А сейчас считайте, что это выражение с круглыми скобками, в которых можно указывать параметры. В результате вычисления функции получается некоторое значение.

Функция **parseInt(cipOKa,** основание) преобразует указанную в параметре строку в целое число в системе счисления по указанному основанию (8, 10 или 16). Если основание не указано, то предполагается 10, то есть десятеричная система счисления.

Примеры

```
parseInt("3.14")          // результат= 3
parseInt("-7.875")        // результат = -7
parseInt("435")           // результат=435
parseInt("Вася")          // результат = NaN, то есть не является числом
parselnt ("15" ,8)        // результат= 13
parse!nt ("0xFF",16)      // результат=255
```

Обратите внимание, что при преобразовании в целое число округления не происходит: дробная часть просто отбрасывается.

Функция **parseFloat(cipoKa)** преобразует указанную строку в число с плавающей разделительной (десятичной, основание) точкой.

Примеры

```
parseFloat ("3.14")       // результат= 3.14
parseFloat("-7.875")      // результат = -7.875
parseFloat  ("435")       // результат = 435
parseFloat  ("Вася")      // результат = NaN, то есть не является числом
parseFloat  ("17.5")      // результат = 435
```

Задача преобразования чисел в строки возникает реже, чем обратное преобразование. Чтобы преобразовать число в строку, достаточно к пустой строке прибавить это число, то есть воспользоваться оператором сложения « + ». Например, вычисление выражения **""+3.14** даст в результате строку "3.14". Об операторах будет подробно рассказано ниже.

Для определения того, является ли значение выражения числом, служит встроенная функция 1$МаМ(значение). Вычисление этой функции дает результат логического типа. Если указанное значение не является числом, функция возвращает true, иначе — false. Однако здесь понятие «число» не совпадает с понятием «значение числового типа». Функции **isNaNQ** считает числом и данные числового

типа, и строку, содержащую только число. Логические значения также идентифицируются как числа. При этом значению true соответствует 1,азначению false — 0. Таким образом, если **isNaN** возвращает false, то это означает, что значение параметра имеет числовой тип, либо является числом, преобразованным в строковый тип, либо является логическим (true или false).

Примеры

```
isNaN(123)            // результат false (то есть это - число)
isNaN("123")          /* результат false (то есть это - число,
                         хотя и в виде строки) */
i sNaN ("50 рублей")  // результат true (то есть это - не число)
isNaN(true)           // результат false
isNaN(false)          // результат false
isNaN("Вася")         // результат true (то есть это - не число)
```

В заключение данного раздела рассмотрим так называемые служебные символы, которые можно вставлять в строки.

Символ	Описание
\п	Новая строка
\t	Табуляция
\f	Новая страница
\Ь	Забой
\r	Возврат каретки

Эти символы обычно используются при формировании строковых данных для их последующего отображения. Например, если мы хотим, чтобы сообщение, выводимое на экран в браузере Internet Explorer с помощью функции **alert(),** отображалось в виде нескольких строк, то следует использовать служебный символ п\ (рис. 1.9):

а!егП"Фамилия - ИвановХпИмя - ИванХпОтчество - Иванович")



Рис. 1.9. Окно, создаваемое функцией alertQ, в случае когда строка сообщения содержит два символа п\

Иногда требуется отобразить символы, имеющие служебное назначение. Как, например, отобразить кавычки, если они используются для задания строки символов? Для этой цели используется « \ » (обратная косая черта). Например, чтобы отобразить строку Акционерное общество "Рога и копыта" вместе с кавычками, следует написать такую строку: "Акционерное общество \" **Рога** и **копыта** \"". Обратная косая черта указывает, что следующий непосредственно за ней символ не нужно интерпретировать как символ синтаксиса языка. В нашем примере она показывает, что кавычки не являются признаком начала или окончания строковых дан-

ных, а являются просто элементом этих данных. В Internet Explorer выполнение выражения **a1eИ:("Акционерное** общество \"Рога и копыта\"") даст результат, показанный на рис. 1.10.



Рис. 1.10. Выполнение выражения a!er1:("Акционерное общество \"**Рога** и **копыта\"")**

Заметим, что эту же задачу можно решить и несколько иначе, используя кавычки различных видов (двойные и одинарные). Во-первых, можно написать так: 'Акционерное общество "Рога и **копыта"'.** В этом случае мы использовали одинарные кавычки в качестве признаков начала и конца всей строки. Во-вторых, можно поменять местами кавычки различных видов: "Акционерное общество **'Рога** и копыта'". Однако в этом случае название акционерного общества будет отображаться в одинарных кавычках. Наконец, можно внешние кавычки оставить двойными, а внутренние одинарные кавычки продублировать: "Акционерное общество "Рога и копыта"". Тогда при отображении строки внутренние кавычки будут заменены на двойные.

Неправильное использование кавычек довольно часто вызывает проблемы у новичков.

ВНИМАНИЕ

Кавычки, обрамляющие строковые данные, должны быть одного вида и использоваться парами.

Интерпретатор, обнаружив в тексте программы кавычки, будет искать еще кавычки такого же вида, считая все находящееся между ними строковыми данными.

ВНИМАНИЕ

Внутри строки, заключенной в кавычки одного вида, можно использовать кавычки другого вида (иначе интерпретатор либо выдаст сообщение об ошибке, либо неправильно воспримет данные).

1.4. Переменные и оператор присвоения

Что произойдет, если в программе просто написать данные какого-либо типа, например число 314? Интерпретатор выполнит эту запись, разместив ее во внутреннем формате где-то в памяти компьютера. Вот и все. Чтобы сохранять данные в памяти и в то же время оставлять их доступными для дальнейшего использования, в программах используются переменные.

1.4.1. Имена переменных

Переменную можно считать контейнером для хранения данных. Данные, сохраняемые в переменной, называют значениями этой переменной. Переменная имеет имя — последовательность букв, цифр и символа подчеркивания без пробелов и знаков препинания, начинающуюся обязательно с буквы или символа подчеркивания.

Примеры правильных имен переменных: myFamily, my_adress, _x, tel412_3456.

Примеры неправильных имен переменных: 512group, myadress, tel:412 3456.

При выборе имен переменных нельзя использовать ключевые слова, то есть слова, используемые в определениях конструкций языка. Например, нельзя выбирать слова var, if, else, const, true, false, function, super, switch и ряд других. Список ключевых слов приведен в разделе 1.11. Имя должно отражать содержание переменной. Если имя состоит из нескольких слов, то между ними можно вводить символ подчеркивания или писать их слитно, начиная каждое слово с прописной буквы. Вот примеры: my_first_adress,myFirstAdress. Иногда в качестве первого символа имени используют букву, указывающую на тип данных (значений) этой переменной: с — строковый (character), n — числовой (number), b —логический (boolean), о — объект (object), а — массив (array). Например cAdress, nCena, aMonth.

JavaScript является регистрозависимым языком. Это означает, что изменение регистра символов (с прописных на строчные и наоборот) в имени переменной приводит к другой переменной. Например: variable, Variable и vaRiabLe — различные переменные.

СОВЕТ

Выработайте для себя правила образования имен, которые не должны противоречить указанным выше требованиям. Если им следовать, то уменьшится вероятность ошибок в ваших программах.

1.4.2. Создание переменных

Создать переменную в программе можно несколькими способами:

- С помощью оператора присвоения значений в формате:

  имя_переменной = значение

  Оператор присвоения обозначается символом равенства «=».

  Пример:

  ```
  myName = "Иван"
  ```

- С помощью ключевого слова var (от variable — переменная) в формате:

  var имя_переменной

  В этом случае созданная переменная не имеет никакого значения, но может его получить в дальнейшем с помощью оператора присвоения.

  Пример:

  ```
  var myName
  myName = "Иван"
  ```

- С помощью ключевого слова var и оператора присвоения в формате:

```
var имя_переменной = значение
```

Пример

```
var myName = "Иван"
```

Строка кода программы с ключевым словом var задает так называемую' инициализацию переменной и для каждой переменной используется один раз. Тип переменной определяется типом значения, которое она имеет. В отличие от многих других языков программирования, при инициализации переменной не нужно описывать ее тип. Переменная может иметь значения различных типов и неоднократно их изменять. Например, вы можете написать следующий код программы:

```
var x = "Иван"
// некоторый код
x = "Анна"
// некоторый код
x = 2.5
```

Одно ключевое слово var можно использовать для инициализации сразу нескольких переменных, как с оператором присвоения, так и без него. При этом переменные и выражения с операторами присвоения разделяются запятыми, например:

```
var name = "Вася", address, x = 3.14
```

Если переменная в данный момент имеет значение числового типа, то говорят, что это числовая переменная. Аналогично можно говорить о строковых, логических (булевских), неопределенных (в случае типа null) переменных.

Выше мы использовали оператор присвоения значения переменной, обозначаемый символом равенства «=». Не следует путать этот оператор с отношением равенства и соответствующей операцией сравнения. Выражение с оператором «=» интерпретатор вычисляет следующим образом: переменной слева от него присваивается значение, расположенное справа от него.

Если x и y — две переменные, то выражение x = y интерпретируется так: переменной x присваивается значение переменной y. Иначе говоря, переменной можно присвоить значение другой переменной, указав справа от оператора «=» ее имя. Таким образом, к значениям можно получать доступ опосредованно — через имена переменных. В следующих разделах мы узнаем, что справа от оператора присвоения можно писать не только значения и имена переменных, но и целые выражения. В подразделе 1.5.3 будут рассмотрены дополнительные операторы присвоения, такие как +=, -= и т. п.

1.4.3. Область действия переменных

Переменные, которые созданы в программе с помощью оператора присвоения с использованием ключевого слова var или без него, являются глобальными. Это означает, что они доступны всюду в этой же программе, а также в вызываемых программах из других файлов. Эти переменные также доступны внутри кода функций. Функциям в нашей книге посвящен специальный раздел.

Кроме глобальных существуют и локальные переменные. Они могут быть созданы внутри кода функций. Вы можете определить переменные с одинаковыми на-

званиями и во внешней программе, и в коде функции. В этом случае переменные, определенные в коде функции с помощью ключевого слова var, будут локальными: изменения их значений внутри функции никак не отразятся на переменных с такими же именами, определенных во внешней программе. При этом значения локальных переменных не доступны из внешней программы.

Мы еще неоднократно будем говорить об области действия переменных. Здесь лишь отметим, что в программировании это понятие играет очень важную роль. Нередко область действия называют областью видимости. Переменная может быть видна или не видна внутри программной единицы (функции, подпрограммы). Область видимости, доступности или действия переменной — эквивалентные термины. Кроме них еще используют понятие времени жизни переменной. Время жизни переменных в JavaScript определяется интервалом времени от загрузки до выгрузки программы из памяти компьютера. Так, если программа (сценарий) записаны в HTML-коде веб-страницы, то после его выгрузки весь сценарий вместе с определенными в нем переменными прекращает активное существование.

1.5. Операторы

Операторы предназначены для составления выражений. Оператор применяется к одному или двум данным, которые в этом случае называются операндами. Например, оператор сложения применяется к двум операндам, а оператор логического отрицания — к одному операнду. Элементарное выражение, состоящее из операндов и оператора, вычисляется интерпретатором и, следовательно, имеет некоторое значение. В этом случае говорят, что оператор возвращает значение. Например, оператор сложения, примененный к числам 2 и 3, возвращает значение 5. Оператор имеет тип, совпадающий с типом возвращаемого им значения. Поскольку элементарное выражение с оператором и операндами возвращает значение, это выражение можно присвоить переменной.

Один оператор мы уже рассмотрели в предыдущем разделе. Это оператор присвоения «=». Мы часто будем его использовать. Однако следует заметить, что существует еще пять разновидностей оператора присвоения, сочетающих в себе действия обычного оператора «=» и операторов сложения, вычитания, умножения, деления и деления по модулю. Эти дополнительные операторы присвоения мы рассмотрим позже в подразделе 1.5.3.

1.5.1. Комментарии

Начнем с операторов комментария. Они позволяют выделить фрагмент программы, который не выполняется интерпретатором, а служит лишь для пояснений содержания программы.

В JavaScript допустимы два вида операторов комментария:

- // — одна строка символов, расположенная справа от этого оператора, считается комментарием;
- /*...*/ — все, что заключено между/* и */, считается комментарием; с помощью этого оператора можно выделить несколько строк в качестве комментария.

СОВЕТ

Не пренебрегайте комментариями в тексте программ. Это поможет при их отладке и сопровождении. На этапе разработки лучше сначала превратить ненужный фрагмент программы в комментарий, чем просто удалить (а вдруг его придется восстанавливать?).

Даже опытные программисты, возвращаясь к работе над своей программой через месяц, с большим трудом вспоминают, что кчему. В примерах программ мы часто будем использовать комментарии.

1.5.2. Арифметические операторы

Арифметические операторы, такие как сложение, умножение и т. д., в JavaScript могут применяться к данным любых типов. Что из этого получается, мы рассмотрим немного позже. А сейчас просто перечислим их (табл. 1,2).

Таблица 1.2. Арифметические операторы

Оператор	Название	Пример
+	Сложение	X+Y
-	Вычитание	X-Y
*	Умножение	X* Y
/	Деление	X/Y
%	Деление по модулю	X%Y
++	Увеличение на 1	X++
—	Уменьшение на 1	Y-

Арифметические операторы лучше всего понять на примере чисел. Первые четыре оператора все проходили в начальных классах. Оператор деления по модулю возвращает остаток отделения первого числа на второе. Например, 8%3 возвращает 2. Операторы «++» и «-» сочетают в себе действия операторов соответственно сложения и вычитания, а также присвоения. Выражение X++ эквивалентно выражению X+1, а выражение X— эквивалентно выражению X-1.

Операторы «++» и «--» называют соответственно инкрементом и декрементом.

Символ «-» используется не только как бинарный (для двух операндов) оператор сложения, но и как унарный (для одного операнда) оператор отрицания для указания того, что число является отрицательным.

Примеры

```
y = 5          // значение переменной y равно 5
x = y + 3      // значение переменной x равно 8 (5+3)
x++            // значение переменной x стало равным 9 (8+1)
x--            // значение переменной x стало равным 8 (9-1)
y++            // значение переменной y равно 6 (5+1)
5-3            // значение равно 2
-3             // отрицательное число
```

Однако формально ничто не мешает нам применить эти операторы к данным других типов. В случае строковых данных оператор сложения дает в результате стро-

ку, полученную путем приписывания справа второй строки к первой. Например, выражение **"Вася"+** "Маша" **возвращает** в результате **"ВасяМаша". Поэтому** для строк оператор сложения называется оператором склейки или конкатенации. Применение остальных арифметических операторов к строкам дает в результате NaN - значение, не являющееся числом (подробности см. в подразделе 1.7.3. В случае, когда оператор сложения применяется к строке и числу, интерпретатор переводит число в соответствующую строку и далее действует как оператор склейки двух строк.

Примеры

```
x = "Вася"          // значение переменной x равно "Вася"
y = "Маша"          // значение переменной y равно "Маша"
z = x + " " +y      // значение переменной z равно "Вася Маша"
z = x + 5           // значение переменной z равно "Вася5"
п = "20" + 5        /* значение переменной п равно "205",
                       а не 30 или "30" */
```

В случае логических данных интерпретатор переводит логические значения операндов в числовые (true в 1, **false** в 0), выполняет вычисление и возвращает числовой результат. То же самое происходит в том случае, когда один оператор логический, а другой — числовой.

Примеры

```
true + true        //возвращает2
true + false       // возвращает 1
true * true        //возвращает 1
true/false         /* возвращает Infinity (Неопределенность,
                      так как на ноль делить нельзя) */
true + 5           //возвращает 6
false + 5          //возвращает 5
true * 5           // возвращает 5
true / 5           //возвращает 0.2
```

Если один операнд строкового типа, а другой — логического, то в случае сложения интерпретатор переводит оба операнда в строковый тип и возвращает строку символов, а в случае других арифметических операторов он переводит оба операнда в числовой тип.

Примеры

```
"Вася" + true      // возвращает "Bacfltrue"
"5" + true         // возвращает "Strue"

"Вася" " true      /* возвращает NaN (то есть значение,
                      не являющееся числом) */
"5" * true         // возвращает 5
"5" * false        // возвращает 0
"5" / true         // возвращает 5
```

На первый взгляд, применение арифметических операторов к разнотипным данным может показаться довольно запутанным. Поэтому советую новичкам, особенно на первых порах, руководствоваться следующими простыми правилами:

- применяйте арифметические операторы к данным одного и того же типа;
- оператор сложения применительно к строкам действует как оператор их склейки (приписывания, конкатенации);

- в случае применения арифметических операторов к логическим данным интерпретатор рассматривает значения true и false как числа соответственно 1 и О, и возвращает результат в числовом виде.

Кроме этих правил нужно помнить, что в JavaScript имеются средства для преобразования типов данных, то есть для преобразования данных одного типа в данные другого типа. К этому вопросу мы вернемся позже, а сейчас рассмотрим другие операторы.

1.5.3. Дополнительныеоператоры присвоения

В начале этого раздела мы уже говорили, что кроме обычного оператора присвоения «=» имеются еще пять дополнительных операторов, сочетающих в себе действия обычного оператора присвоения и арифметических операторов.

Оператор	Пример	Эквивалентное выражение
+=	X+-Y	X = X + Y
-=	X-=Y	X = X - Y
*–	X*—Y	X - X * Y
/ -	X/=Y	X = X/Y
% =	X % = Y	x = X % = Y

Поскольку действие арифметических операторов и обычного оператора присвоения уже известны, дополнительные операторы присвоения мы не будем здесь подробно рассматривать. Обратим лишь внимание на экономный способ записи выражений, предоставляемый этими операторами.

Пример:

```
x = "Вася"
x+= ", привет" // "Вася, привет"
```

1.5.4. Операторысравнения

В программах часто приходится проверять, выполняются ли какие-либо условия. Например, на веб-страницах иногда проверяется, является ли браузер пользователя Microsoft Internet Explorer или Netscape Navigator. В зависимости от результата процесс дальнейших вычислений может пойти по тому или другому пути. Проверяемые условия формируются на основе операторов сравнения, таких как «равно», «меньше», «больше» и т. п. Результатом вычисления элементарного выражения, содержащего оператор сравнения и операнды (сравниваемые данные), является логическое значение, то есть true или false. Так, если условие выполняется (верно, справедливо), то возвращается true. В противном случае возвращается false.

Оператор	Название	Пример
= =	Равно	X = =Y
[=	Не равно	X I- Y

продолжение iK

Оператор	Название	Пример
>	Больше, чем	X*Y
>=	Больше или равно (не меньше)	X>=Y
<	Меньше, чем	X<Y
<=	Меньше или равно (не больше)	X <= Y

ВНИМАНИЕ

Оператор «равно» записывается с помощью двух символов без пробелов между ними.

Сравнивать можно числа, строки и логические значения. Сравнение чисел происходит по правилам арифметики, а строк — путем сравнения ASCII-кодов символов начиная с левого конца строк. Логические значения сравниваются так же, как и числа 1 и 0 (true соответствует 1, a false — 0). Как видим, с числами и логическими данными все довольно просто. А вот результат сравнения строк не всегда очевиден.

Примеры

```
"abcd"=="abc"         // возвращает false
"abc"=="abcd"         // возвращает false
"abcd"=="abcd"        // возвращает true
"abcd"==" abed"       /* возвращает false (1-й символ
                         2-го операнда - пробел) */
"abed" > " abed"      // возвращает true
"abc" < "abed"        // возвращает true
"235ab" < "abcdxyz"   // возвращает true
"235xyz" < "abc"      // возвращает true
```

Мы не приводим здесь таблицу кодов ASCII для всех символов. Отметим лишь, что некоторые из них упорядочены в порядке возрастания ASCII-кодов следующим образом: сначала идет пробел, затем в порядке возрастания цифры от 0 до 9, символ подчеркивания, латинские и кириллические буквы по алфавиту (сначала прописные, а затем строчные).

Заметим, что операторы сравнения могут быть применены и к разнотипным данным. Если сравниваются строка и число, то интерпретатор приводит операнды к числовому типу. То же самое происходит при сравнении логических данных и числа. Если сравниваются логические данные и строка, то дело обстоит несколько сложнее. В этом случае результат не зависит от содержимого строки. Если она содержит число (но не цифры с буквами), только пробелы или является пустой, то операнды приводятся к числовому типу. При этом пустая строка ("") или содержащая только пробелы преобразуется в число 0. В остальных случаях все операторы сравнения, кроме «!=», будут возвращать **false** (а оператор «!=» — противоположный результат, то есть true).

Примеры

```
false < "57"        // возвращает true
false < "-2.5"      // возвращает false
true >= "0.5"       // возвращает true
false < "57ab"      // возвращает false
false > "57ab"      // возвращает false
```

```
true  == "true"          // возвращает false
true  != "true"          // возвращает true
true  <= "true"          // возвращает false
true  >= "true"          // возвращает false
```

1.5.5. Логические операторы

Логические данные, обычно получаемые с помощью элементарных выражений, содержащих операторы сравнения, можно объединять в более сложные выражения. Для этого используются логические (булевские) операторы — логические союзы И и ИЛИ, а также оператор отрицания НЕ. Например, нам может потребоваться сформировать сложное логическое условие следующего вида: «возраст не более 30 и опыт работы больше 10, или юрист». В этом примере есть и операторы сравнения, и логические операторы. Выражения с логическими операторами возвращают значение true или false.

<table>
<tr><th>Оператор</th><th>Название</th><th>Пример</th></tr>
<tr><td>!</td><td>Отрицание (НЕ)</td><td>!X</td></tr>
<tr><td>&&</td><td>И</td><td>X&&Y</td></tr>
<tr><td>||</td><td>ИЛИ</td><td>X||Y</td></tr>
</table>

Оператор отрицания «!» применяется к одному операнду, изменяя его значение на противоположное: если X имеет значение true, то !X возвращает значение false, и наоборот, если X имеет значение false, то !X возвращает значение true.

Ниже в таблице указано, какие значения возвращают операторы И и ИЛИ при различных логических значениях двух операндов.

<table>
<tr><th></th><th>Y</th><th>X&&Y</th><th>X||Y</th></tr>
<tr><td>true</td><td>true</td><td>true</td><td>true</td></tr>
<tr><td>true</td><td>false</td><td>false</td><td>true</td></tr>
<tr><td>false</td><td>true</td><td>false</td><td>true</td></tr>
<tr><td>false</td><td>false</td><td>false</td><td>false</td></tr>
</table>

Операторы «&&» и «||» еще называют логическим умножением и логическим сложением соответственно. Если вспомнить, что значению true можно сопоставить 1, а значению false — 0, то нетрудно понять, как вычисляются значения элементарных выражений с логическими операторами. Нужно только учесть, что в алгебре логики 1 + 1 = 1 (а не 2). Аналогично оператору отрицания соответствует вычитание из единицы числового эквивалента логического значения операнда. В математике логические операции И и ИЛИ называют соответственно конъюнкцией и дизъюнкцией.

СОВЕТ

Чтобы не запутаться, не применяйте логические операторы к нелогическим данным и особенно к разнотипным данным.

Примеры

```
x = false || 2*2 ==4        // значение переменной x равно true
x = 5<2 || "abed" <="xy"    // значение переменной x равно true
y = !x                      // значение переменной y равно false
z = x && y                  // значение переменной z равно false
z = x || y                  // значение переменной z равно true
```

Сложные логические выражения, состоящие из нескольких более простых, соединенных операторами И и ИЛИ, выполняются по так называемому принципу короткой обработки. Дело в том, что значение всего выражения бывает можно определить, вычислив лишь одно или несколько более простых выражений, не вычисляя остальные. Например, выражение x&&y вычисляется слева направо; если значение x оказалось равным false, то значение y не вычисляется, поскольку и так известно, что значение всего выражения равно false. Аналогично если в выражении x || y значение x равно true, то значение y не вычисляется, поскольку уже ясно, что все выражение равно true. Данное обстоятельство требует особого внимания при тестировании сложных логических выражений. Так, если какая-нибудь составная часть выражения содержит ошибку, то она может остаться невыявленной, поскольку эта часть выражения просто не выполнялась при тестировании.

1.5.6. Операторы условного перехода

Вычислительный процесс можно направить по тому или другому пути в зависимости от того, выполняется ли некоторое условие или нет. Этой цели служат операторы условного перехода if и switch.

Оператор if

Оператор условного перехода if позволяет реализовать структуру условного выражения если ..., то ..., иначе ...

Синтаксис оператора if перехода следующий:

```
if (условие)
   { код, который выполняется, если условие выполнено}
else
   { код, который выполняется, если условие не выполнено}
```

В фигурных скобках располагается блок кода — несколько выражений. Если в блоке используется не более одного выражения, то фигурные скобки можно не писать. Часть этой конструкции, определяемая ключевым словом else (иначе), необязательна. В этом случае остается только часть, определенная ключевым словом if (если):

```
if (условие)
      { код, который работает, если условие выполнено}
```

Конструкция оператора условного перехода допускает вложение других операторов условного перехода. Условие обычно представляет собой выражение логического типа, то есть выражение, значение которого есть true или false. Обычно это элементарные выражения с операторами сравнения.

Примеры

1. Выводится диалоговое окно с тем или иным сообщением в зависимости от значения переменной age (возраст).

```
if  (age<18)
      {alertC'Bbi слишком молоды для просмотра этого сайта")}
else
      {alert("Подтвердите свое решение заглянуть на этот сайт")}
```

2. Выводится диалоговое окно с сообщением, если только значение переменной аде меньше 18.

```
if  (age<18)
      {alertC'Bbi слишком молоды для просмотра этого сайта")}
```

Делать ли отступы при написании операторов, где располагать фигурные скобки — дело вкуса. Следует руководствоваться наглядностью и ясностью структуры, при которой легко проверить правильность расстановки скобок. Кроме описанного выше способа, часто встречается еще и такой:

```
if (условие) {
      код. который работает, если условие выполнено
} else {
      код, который работает, если условие не выполнено
}
```

Возможно, этот способ записи лучше отражает структуру оператора условного перехода.

Более сложная структура оператора условного перехода получается при вложении других операторов if:

```
if (условие!) {
         код, который работает, если условие! выполнено
} else { if (условие2) {
         код, который работает, если условие2 выполнено
         }else{
код, который работает, если условие2 не выполнено
         }
}
```

Выше мы уже отмечали, что условие в операторе if обычно является логическим выражением. Однако это может быть также и строковое, и числовое выражение. В случае строкового выражения условие считается выполненным, если его значением является непустая строка. Напомним, что пустая строка "" не содержит ни одного символа, в том числе и пробела (строка, содержащая хотя бы один пробел, не пуста). В случае числового выражения условие считается выполненным, если его значением является число, отличное от нуля. Во многих случаях эта многозначность типа условия оказывается очень удобной.

Допустим, что переменная x содержит данные, которые ввел пользователь, и нам требуется проверить, что он действительно что-то ввел. В следующем примере мы проверяем, что значение переменной x не пусто (не 0, не пустая строка "" и не null). Если x пусто, то выводится соответствующее сообщение:

```
if (!x)  {
      alertC'Bbi ничего не ввели")
}
```

Заметим, что поскольку в этом примере блок кода содержит всего лишь одно выражение, постольку фигурные скобки можно опустить, да и всю конструкцию оператора условного перехода можно записать в одной строке:

```
if (!x) alertC'Bbi ничего не ввели")
```

Оператор switch

Для организации проверки большого количества условий вполне достаточно использовать рассмотренный выше оператор if. Однако в случае нескольких условий более удобным и наглядным оказывается оператор switch (переключатель). Он особенно удобен, если требуется проверить несколько условий, которые не являются взаимоисключающими.

Синтаксис оператора switch выглядит следующим образом:

```
switch (выражение) {
   case вариант1:
      код
      [break]
   case вариант2:
      код
      [break]
   ...
   [default:
код]
}
```

Параметр *выражение* оператора switch может принимать строковые, числовые и логические значения. Разумеется, в случае логического выражения возможны только два варианта. Ключевые слова (операторы) break и default не являются обязательными, что отражено с помощью прямоугольных скобок. Здесь они являются элементами описания синтаксиса, и при написании операторов их указывать не нужно. Если оператор break указан, то проверка остальных условий не производится. Если указан оператор default, то следующий за ним код выполняется, если значение выражения не соответствует ни одному из вариантов. Если все варианты возможных значений предусмотрены в операторе switch, то оператор default можно не использовать.

Оператор switch работает следующим образом. Сначала вычисляется выражение, указанное в круглых скобках сразу за ключевым словом switch. Полученное значение сравнивается с тем, которое указано в первом варианте. Если они не совпадают, то код этого варианта не выполняется и происходит переход к следующему варианту. Если же значения совпали, то выполняется код, соответствующий этому варианту. При этом, если неуказан оператор break, то выполняются коды и остальных вариантов, пока не встретится оператор break. Это же правило действует и для остальных вариантов.

Пример

```
, x = 2
switsch (x) {
      case 1:
         alert(l)
      case 2:
         alert(2)
      case 3:
         alert(3)
}
```

В приведенном примере сработают 2-й и 3-й варианты. Если мы хотим, чтобы сработал только один какой-нибудь вариант (только тот, который соответствует значению выражения), то нужно использовать оператор break.

Пример

```
x = 2
switsch (x) {
    case 1:
        alert (l) ;
        break
    case 2:
        alert(2);
        break
    case 3:
        alert(3) ;
        break
}
```

В этом примере сработает только 2-й вариант.

Пример

Допустим, переменная xlang содержит название языка, который выбрал пользователь и ввел в поле формы. Тогда возможен такой вариант применения оператора switch:

```
switsch (xlang) {
    case "английский" :
        window.open ("engl.htm");
        break
    case "французский":
        window.open ("french .htm") ;
        break
    case "немецкий" :
        window.open("german.htm") ;
        break
    default:
        alert("У нас нет документа на таком языке")
/
```

Здесь выражение window.openQ открывает новое окно браузера и загружает в него указанный в скобках HTML-документ. Заметим, что эту же задачу можно решить и с помощью вложенных операторов if:

```
if (xlang == "английский")
    window. open ("engl .htm")
else {
    if (xlang == "французский")
        window. open ("french .htm") ;
    else {
        if (xlang == "немецкий")
            window.open ("german.htm")
    /   else
            alertC'V нас нет документа на таком языке");
        }
    }
```

Для тренировки перепишем приведенный выше код в другом виде:

```
if (xlang == "английский") window.open("engl.htm")
    else { if (xlang == "французский") window. open("french. htm")
    else { if (xlang == "немецкий") wi ndow. open ("german . htm")
    else alert tC'Үнас нет документа на таком языке")
        }
        }
```

1.5.7. Операторы цикла

Оператор цикла обеспечивает многократное выполнение блока программного кода до тех пор, пока не выполнится некоторое условие. В JavaScript предусмотрены три оператора цикла: for, while и do-while. Вообще говоря, при создании программ вполне можно обойтись одним из них, for или while. Мне, например, больше нравится while. Однако возникают ситуации, в которых один из операторов более удобен или естествен, чем другой.

Оператор for

Оператор for (для) также называют оператором со счетчиком циклов, хотя в нем совсем не обязательно использовать счетчик. Синтаксис этого оператора следующий:

```
for ( [начальное выражение] ; [условие] ; [выражение обновления] )
{
        код
}
```

Здесь квадратные скобки лишь указывают на то, что заключенные в них параметры не являются обязательными. Как и в случае оператора условного перехода, возможна и такая запись:

```
for ( [начальное выражение] ; [условие] ; [выражение обновления] ) {
        код
}
```

Все, что находится в круглых скобках справа от ключевого слова for, называется заголовком оператора цикла, а содержимое фигурных скобок — его телом.

В заголовке оператора цикла начальное выражение выполняется только один раз в начале выполнения оператора. Второй параметр представляет собой условие продолжения работы оператора цикла. Он аналогичен условию в операторе условного перехода if. Третий параметр содержит выражение, которое выполняется после выполнения всех выражений кода, заключенного в фигурные скобки.

Оператор цикла работает следующим образом. Сначала выполняется *начальное выражение.* Затем проверяется *условие.* Если оно выполнено, то оператор цикла прекращает работу (при этом код не выполняется). В противном случае выполняется код, расположенный в теле оператора for, то есть между фигурными скобками. После этого выполняется выражение обновления (третий параметр оператора for). Таким образом, заканчивается первый цикл или, как еще говорят, первая итерация цикла. Далее снова проверяется условие, и все повторяется описанным выше способом.

Обычно в качестве начального выражения используют оператор присваивания значения переменной. Например, i = 0 или var i = 0. Имя переменной и присваиваемое значение могут быть любыми. Эту переменную называют счетчиком циклов. В этом случае условие, как правило, представляет собой элементарное выражение сравнения переменной счетчика циклов с некоторым числом, например, i<=nMax. Выражение обновления в таком случае просто изменяет значение счетчика циклов, например i = i + 1 или, короче, i++.

В следующем примере оператор цикла просто изменяет значение своего счетчика, выполняя 15 итераций:

```
for (i = 1 ; i <= 15; i++) {}
```

Немного модифицируем этот код, чтобы вычислить сумму всех целых положительных чисел от 1 до 15:

```
var s=1
for (i = 1; i <= 15; i++) {
   s = s + i
}
```

Заметим, что счетчик циклов может быть не только возрастающим, но и убывающим.

Пример

Допустим, требуется вычислить x в степени y, где x, y — целые положительные числа. Алгоритм решения этой задачи прост: надо вычислить выражение x*x*x* ... x, в котором сомножитель x встречается y раз. Очевидно, что если y не превышает 10, то можно воспользоваться оператором умножения: выражение будет не очень длинным. А как быть в том случае, когда y равно нескольким десяткам или сотням? Ясно, что в общем случае следует воспользоваться оператором цикла:

```
/* Вычисляем x в степени y */
var z = x                                  // z хранит результат
for (i = 2 ; i <= y ; i++) {
      z=z*x
}
```

В этом примере результат сохраняется в переменной z. Начальное ее значение равно x. Если y равно 1, то оператор цикла не будет выполняться, поскольку не выполняется условие 2 <= 1 (обратите внимание на начальное значение счетчика циклов), — и, следовательно, мы получим верный результат: x в степени 1 равно x. Если y равно 2, то оператор цикла выполнит одну итерацию, вычислив выражение z = z*x при z, равном x (то есть z = x*x). При y, равном 3, оператор цикла сделает две итерации. На второй, последней итерации выражение z = z*x вычисляется при текущем значении z, равном x*x, и, следовательно, становится равным x*x*x (то - есть x в степени 3).

Пример

Рассмотрим довольно традиционный при изучении операторов цикла пример вычисления факториала числа. Факториал числа n в математике обозначают как n!. Для n, равного 0 и 1, n! равен 1. В остальных случаях n! равен 2*3*4* ... *n. Поскольку возможны два варианта исходных данных, нам потребуется использовать оператор условного перехода. Код, решающий эту задачу, выглядит следующим образом:

```
/* Вычисляем n! */
var z = 1                                  // z хранит результат n!
if (n >1) {
      for (i = 2 ; i <= n ; i++) {
            z=z*i
      }
}
```

Для принудительного (то есть не по условию) выхода из цикла используется оператор break (прерывание). Если вычислительный процесс встречает этот оператор в теле оператора цикла, то он сразу же завершается без выполнения последующих выражений кода в теле и даже выражения обновления. Обычно оператор

break применяется при проверке некоторого дополнительного условия, выполне-' ние которого требует завершения цикла, несмотря на то что условие в заголовке цикла еще не выполнено. Типовая структура оператора цикла с использованием break имеет следующий вид:

```
for ( [начальное выражение] ; [условие!] ; [выражение обновления] )
{
        код
        if (условие2){
            код
            break
        }
        код
}
```

Для управления вычислениями в операторе цикла можно также использовать оператор continue (продолжение). Также, как и break, этот оператор применяется в теле оператора цикла вместе с оператором условного перехода. Однако, в отличие от break, оператор continue прекращает выполнение последующего кода, выполняет выражение обновления и возвращает вычислительный процесс в начало оператора цикла, где производится проверка условия, указанного в заголовке. Типовая структура оператора цикла с использованием break имеет следующий вид:

```
for ( [начальное выражение] ; [условие!] ; [выражение обновления] )
{
        код
        if (условие2){
        код
        continue
        }
        код
}
```

Оператор while

Оператор цикла while (до тех пор, пока) имеет структуру более простую, чем оператор for, и работает несколько иначе. Синтаксис этого оператора следующий:

```
while ( условие )
{
        код
}
```

При выполнении этого оператора сначала производится проверка условия, указанного в заголовке, то есть в круглых скобках справа от ключевого слова while. Если оно выполняется, то выполняется код в теле оператора цикла, заключенного в фигурные скобки. В противном случае код не выполняется. При выполнении кода (завершении первой итерации) вычислительный процесс возвращается к заголовку, где снова проверяется условие, и т. д.

Если сравнивать оператор while с оператором for, то особенность первого заключается в том, что выражение обновления записывается в теле оператора, а не в заголовке. Часто забывают указать это выражение, и в результате цикл не завершается (программа «зависает»).

Рассмотрим несколько примеров решения задач, которые мы уже решали ранее с использованием оператора цикла for.

Пример

Возведение x в степень y.

```
/* Вычисляем x в степени y */
var z = x                                  // z хранит результат
i = 2
while (i = 2) {
        z= z*x
        i++
}
```

Пример

Вычисление п!.

```
/* Ввычисляем п! */
var z = 1                                  // z хранит результат п!
if (n >1) {
    i = 2
    while ( i <= n ) {
            z=z*i
            i++
    ]
]
```

Во всех приведенных выше примерах использовался счетчик циклов. Однако он инициировался заранее, до оператора цикла. Обновление значения этого счетчика производилось в теле оператора цикла.

Для управления вычислительным процессом в операторе while, также как и в операторе for, можно использовать операторы прерывания break и продолжения continue.

ВНИМАНИЕ

Если в while вы используете счетчики циклов, то будьте осторожны, применяя break и continue.

Оператор do-while

Оператор **do-while** (делай до тех пор, пока) представляет собой конструкцию из двух операторов, используемых совместно. Синтаксис этой конструкции следующий:

```
do {
    код
.1
while (условие)
```

В отличие от оператора while в операторе do-while код выполняется хотя бы один раз, независимо от условия. Условие проверяется после выполнения кода. Если оно истинно, то снова выполняется код в теле оператора do. В противном случае работа оператора do-while завершается. В операторе while условие проверяется в первую очередь, до выполнения кода в теле. Если при первом обращении к оператору while условие не выполняется, то код не будет выполнен никогда.

Рассмотрим несколько примеров решения задач, которые мы уже решали ранее с использованием операторов цикла for и while.

Пример

Возведение x в степень y.

```
/* Вычисляем x в степени y */
var z = x                                   // z хранит результат
i = 2
do{
    z=z*x
    i++
}
while (i <= y)
```

Пример

Вычисление n!.

```
/* Вычисляем n! */
var z = 1                                   // z хранит результат n!
if (n >1) {
    i = 2
    do {
        z=z*i
        i++
    }
    while ( i <= n )
}
```

1.5.8. Выражения с операторами

Выше неоднократно встречались термины «выражение» и «элементарное выражение». Вы, должно быть, уже составили себе некоторое представление о том, что такое выражение. Тем не менее уточним его.

Начнем с того, что просто данные (конечные значения) являются выражениями языка. Например, число 5.2, одиноко стоящее в строке текста программного кода, является выражением. Последовательность символов, заключенная в кавычки (например, "Вася"), — тоже выражение. Имя переменной — еще один вариант выражения.

Запись, содержащая имя переменной, за которой следуют символы оператора присвоения и некоторое значение (например, x = "Привет всем!"), является выражением JavaScript. Запись, состоящая из операндов и оператора (например, x + 5), также является выражением.

Все перечисленные выше варианты выражения назовем элементарными выражениями. Тогда записи, содержащие операторы и операнды в виде элементарных выражений, являются выражениями. Например, пусть имеются два элементарных выражения: x+5 и y-3. Тогда запись x+5 + y-3, объединяющая их оператором сложения, является выражением. Это касается не только арифметических операторов, но и всех других.

Итак, мы можем писать сложные выражения, в которых операторы различных типов могут встречаться несколько раз. Выражение с последовательностью из нескольких операторов вычисляется слева направо, но с учетом так называемого приоритета операторов.

ВНИМАНИЕ

Среди арифметических операторов наибольшим приоритетом обладают операторы умножения и деления (в том числе и деления по модулю). Затем следуют сложение и вычитание. Среди логических операторов наибольшим приоритетом обладает отрицание, затем следует логическое И, а дальше — логическое ИЛИ. Операторы сравнения по приоритету выше логических операторов. Последовательность операторов с одинаковым приоритетом выполняется слева направо.

Примеры

```
2 + 3 * 5                    // результат равен 17, а не 25
2 <3 || 3 <1                 // результат равен true
2 <3 || 3 <1 && false        // результат равен true
! 2 <3 || 3 <1 && false      // результат равен false
```

Процессом вычисления выражений можно управлять с помощью круглых скобок. Каждой открывающейся скобке соответствует своя закрывающаяся скобка, так что выражения, заключенные в круглые скобки, могут входить в состав других выражений, которые, в свою очередь, также могут быть заключены в скобки. Таким образом, с помощью круглых скобок можно создать иерархическую структуру выражения, состоящего из других выражений. Первыми выполняются выражения, имеющие наибольшую глубину вложенности. Далее выполняются выражения, находящиеся на меньшей глубине в иерархии, и т. д. Выражения с одинаковой глубиной вложенности выполняются в порядке, принятом по умолчанию, то есть слева направо.

ВНИМАНИЕ

В выражении с круглыми скобками количество открывающихся скобок должно быть равно количеству закрывающихся, а общее количество всех круглых скобок должно быть четным числом. Если это не выполняется, то интерпретатор выдает сообщение о синтаксической ошибке. Ошибки такого рода часто встречаются при программировании.

Примеры

```
2 + 3 * 5                    // результат равен 17
(2 + 3) * 5                  // результат равен 25
((2 +3) + 4*5)/2             // результат равен 12.5
(2 + 3) + 4*5/2              // результат равен 15
2 + 3 + 4*5/2                // результат равен 15
(2 + 3 + 4)*5/2              // результат равен 22.5
2 + (3 + 4*5)/2              // результат равен 13.5
(2 + (3 + 4)*5)/2            // результат равен 18.5
```

Операторы условного перехода и цикла также представляют собой выражения языка JavaScript. Кроме обычных способов их записи, рассмотренных выше, принципиально возможна запись в одну строку. При этом необходимо, чтобы выражения в блоках кода разделялись точкой с запятой.

Пример

```
if (!x){x = "Вы ничего не ввели"; alert(x) }else alert("Все в порядке")
```

Операторы условного перехода и цикла возвращают значения подобно другим операторам, а именно возвращают значение последнего выполненного выражения.

Сводные сведения о приоритетах всех операторов приведены в разделе 1.10 в конце данной главы.

1.6. Функции

Функция представляет собой подпрограмму, которую можно вызвать для выполнения, обратившись к ней по имени. Взаимодействие функции с внешней программой, из которой она была вызвана, происходит путем передачи функции параметров и приема от нее результата вычислений. Впрочем, функция в JavaScript может и не требовать параметров, а также ничего не возвращать.

В JavaScript есть встроенные функции, которые можно использовать в программах, но код которых нельзя редактировать или посмотреть. Все, что мы можем узнать о них, — это описание ихдействия, параметров и возвращаемого значения.

Кроме использования встроенных функций вы можете создать свои собственные, так называемые пользовательские функции. Часто используемые фрагменты программного кода целесообразно оформлять в виде функций. Такой фрагмент кода заключается в фигурные скобки, а перед ним пишется ключевое слово function, за которым следуют круглые скобки, обрамляющие список параметров. Более подробно пользовательские функции будут рассмотрены в подразделе 1.6.2.

Чтобы вызвать функцию в программе, следует написать выражение в следующем формате:

```
имя_функции (параметры)
```

Если требуются параметры, то они указываются в круглых скобках через запятую. Функция может и не иметь параметров. В этом случае в круглых скобках ничего не указывается. Подробности использования функций изложены далее в этом разделе.

1.6.1. Встроенные функции

В JavaScript имеются нижеследующие встроенные функции (некоторые из них мы уже рассматривали ранее). Хотя для иллюстрации работы этих функций приводится множество примеров, желательно выполнить их самим, а также придумать свои примеры, обращая внимание на крайние (даже абсурдные) случаи.

parseInt(строка, основание) — преобразует указанную строку в целое число в системе счисления по указанному основанию (8,10 или 16); если основание не указано, то предполагается 10, то есть десятеричная система счисления.

Примеры

```
parseInt("2.5")                  // результат = 2
parseInt("-17.875")              // результат = -17
parseInt("1952")                 // результат = 1952
parseInt("150 руб.")             // результат = 150
parseInt("цена 150 руб.")        // результат = NaN
```

parseFloat(строка, основание) — преобразует указанную строку в число с плавающей разделительной (десятичной) точкой в системе счисления по указанному основанию (8,10 или 16); если основание не указано, то предполагается 10, то есть десятеричная система счисления.

Примеры

```
parseFloat("2.5")                // результат = 2.5
parseFloat("-17.875")            // результат = -17.875
```

```
parseFloat ("1952")              // результат = 1952
pareseFloat("1.50px")            // результат = 1.5
pareseFloat("двести")            // результат = NaN
```

isNaN (значение) — возвращает true, если указанное в параметре значение не является числом, иначе — false.

Примеры

```
isNaN(123)                       // результат false
isNaN("123")                     // результат false
isNaN("Ten. 1234567")            // результат true
isNaN("35 px")                   // результат true
isNaN(true)                      // результат false
isNaN(false)                     // результат false
isNaN("Bacfl")                   // результат true
```

Перечисленные выше функции мы рассматривали в разделе 1.3, посвященном типам данных.

eval(сТроКа)—вычисляетвыражениевуказаннойстроке;выражениедолжнобыть написано на языке JavaScript (не содержит тегов HTML).

Примеры

```
var y = 5                        // значение y равно 5
var x = "if(y<10) {y = y+2}"     // значение x равно строке символов
eval(x)                          // значение y равно 7
```

Другой пример применения функции evalQ можно найти в начале данной главы. Это — создание веб-страницы, содержащей текстовое поле для ввода и выполнения выражений JavaScript. Вот текст соответствующего HTML-кода со сценарием, содержащим функцию eval():

```
<HTML>
<TEXTAREA ID = "mycode" ROWS = 10 COLS = 60></TEXTAREA>
<TEXTAREA ID = "myrezult" ROWS = 3 COLS = 60></TEXTAREA>
<P>
<BUTTON onclick ="document.all.myrezult.value=eval(mycode.value)">
Выполнить </BUTTON>
< BUTTON onclick="document.all.mycode.value='';
document.all.myrezult.value=''">
Очистить</ BUTTON >
<P>
</HTML>
```

Здесь сценарии записаны в виде символьных строк в качестве значений атрибутов onclik, определяющих событие щелчок кнопкой мыши на HTML-кнопках, которые заданы тегами <button>.

escape(сгроКа) — возвращает строку в виде %XX, где XX — ASCII-код указанного символа; такую строку еще называют escape-последовательностью.

unescape(сгроКа) — осуществляет обратное преобразование.

При взаимодействии браузеров и серверов протоколы передачи данных позволяют передавать не все символы в их естественном виде. Для передачи остальных символов используются их шестнадцатеричные ASCII-коды, перед которыми указывается символ « %». Например, пробел представляется в escape-последовательности как %20.

Примеры

```
escape("How do you do")          // значение равно "How%20do%20you%20do"
езсареС'Привет") /* значение равно
%u041F%u0440%u0438%u0432%u0435%u0442 */
```

Другие примеры применения функций escape() и unescapeQ приведены в разделе 2.8 (глава 2).

typeOf(с'бъект) — возвращает тип указанного объекта в виде символьной строки,; например "boolean", "function" и т. п.

1.6.2. Пользовательские функции

Пользовательские функции — это функции, которые вы можете создать сами, по своему усмотрению, для решения своих задач. Функция задается своим определением (описанием), которое начинается ключевым словом function. Точнее, описание функции имеет следующий синтаксис:

```
function имя_функции(параметры)
!
\ код
J
```

Часто определение функции записывают и в таких формах:

```
function имя_функции(параметры) {
   код
}

function имя_функции(параметры) { код }
```

Имя функции выбирается также, как и имя переменной. Недопустимо использовать в качестве имени ключевые слова языка JavaScript. За именем функции обязательно стоит пара круглых скобок. Программный код (тело) функции заключается в фигурные скобки. Они определяют группу выражений, которые относятся к коду именно этой функции. Если функция принимает параметры, то список их имен (идентификаторов) указывается в круглых скобках около имени функции. Имена параметров выбираются согласно тем же требованиям, что и имена обычных переменных. Если параметров несколько, то в списке они разделяются запятыми. Если параметры для данной функции не предусмотрены, то в круглых скобках около имени функции ничего не пишут.

Когда создается определение функции, список ее параметров (если он необходим) содержит просто формальные идентификаторы (имена) этих параметров, понимаемые как переменные. В определении функции в списке параметров, заключенном в круглые скобки сразу же за именем функции после ключевого слова function, нельзя использовать конкретные значения и выражения. В этом смысле определение функции задает код, оперирующий формальными параметрами, которые конкретизируются лишь при вызове функции из внешней программы.

Если требуется, чтобы функция возвращала некоторое значение, то в ее теле используется оператор возврата return с указанием справа от него того, что следует возвратить. В качестве возвращаемой величины может выступать любое выражение: простое значение, имя переменной или вычисляемое выражение. Оператор return может встречаться в коде функции несколько раз. Впрочем, возвращаемую

величину, а также сам оператор return можно и не указывать. В этом случае функция ничего не будет возвращать.

Пример

Допустим, требуется определить функцию для вычисления площади прямоугольника (для этого необходимо умножить ширину на высоту). Назовем эту функцию Srectangle. Тогда ее определение будет выглядеть следующим образом:

```
function Srectangle(width, height){
S = width * height
return S
}
```

Здесь сначала вычисляется площадь путем умножения значений параметров width и height, полученное значение присваивается переменной S,азатем оператор return возвращает значение этой переменной.

Определение функции Srectangle можно сделать более экономным, минуя присвоение значения переменной S:

```
function Srectangle(width, height){
return width * height
}
```

Определение функции, описанное выше, будучи помещенным в программу, усваивается интерпретатором, но сама функция (ее код или тело) не выполняется. Чтобы выполнить функцию, определение которой задано, необходимо написать в программе выражение вызова этой функции. Оно имеет следующий синтаксис:

```
имя_функции (параметры)
```

Имя функции должно полностью (вплоть до регистра) совпадать с именем ранее определенной функции. Параметры, если они заданы в определении функции, в вызове функции представляются конкретными значениями, Переменными или выражениями.

ВНИМАНИЕ

Не путайте определение функции с ее вызовом, хотя и то и другое могут находиться в одной и той же программе.

В JavaScript можно не поддерживать равенство между количествами параметров в определении функции и в ее вызове. Если в функции определены, например, три параметра, а в вызове указаны только два, то последнему параметру будет автоматически присвоено значение null. Наоборот, лишние параметры в вызове функции будут просто проигнорированы.

Выше мы рассмотрели пример определения функции **SrectangleQ,** вычисляющей площадь прямоугольника, если заданы его ширина и высота. Чтобы вычислить значение площади прямоугольника при конкретных значениях параметров, в программе необходимо написать выражение вызова функции **SrectangleQ,** указав в качестве ее параметров конкретные значения, либо переменные с конкретными значениями, либо выражения, вычисляющие значения.

Примеры

```
Srectangle(3, 5)            /* возвращает площадь прямоугольника
                               размером 3x5 */
```

```
Srectangle(3, 4 + 2)            /* возвращает площадь прямоугольника
                                   размером 3x6 */

height = 8
Srectangle(3, height)           /* возвращает площадь прямоугольника
                                   размером 3x8 */

width=4
height=5
Srectangle(widt, height + 2)    /* возвращает площадь прямоугольника
                                   размером 4x7 */
Srectangle(2)                   /* возвращает 0 (второй параметр
                                   не задан) */
```

Внутри тела функции можно создавать переменные, либо просто с помощью оператора присвоения, либо с помощью ключевого слова var. При этом возникает вопрос об области действия переменных.

Если вы используете просто оператор присвоения, то возможны две ситуации.

1. Переменная в операторе присвоения встречается первый раз в вашей программе именно в теле функции. В этом случае она действует в пределах кода (тела) этой функции, но после выполнения функции эта переменная продолжает существовать и во внешней программе. Таким образом, эта переменная является глобальной.
2. Переменная в операторе присвоения уже определена во внешней программе. В этом случае она и действует внутри функции, и продолжает существовать после завершения ее выполнения, поскольку она была создана во внешней программе как глобальная переменная.

Если в теле функции вы используете ключевое слово var для инициализации переменной, то эта переменная будет действовать только в пределах кода функции, независимо от того, была ли она определена во внешней программе или нет. При этом если переменная создана с ключевым словом var впервые в теле функции, то она является локальной, то есть недоступной из внешней программы. Таким образом, инициализация переменной с помощью выражения с ключевым словом var создает локальную переменную.

Если в теле функции вы используете переменную, определенную только во внешней программе, то изменение ее значения в теле функции сохранится даже после завершения выполнения этой функции, поскольку это глобальная переменная.

Если во внешней программе вы определили некоторые переменные, имена которых совпадают с формальными параметрами в определении функции, а затем указываете их в качестве параметров вызова этой функции, то произойдет следующее. Функция воспримет значения параметров, определенные во внешней программе, однако любые их изменения в теле функции останутся локальными, то есть после завершения работы кода функции значения указанных переменных останутся прежними, как до вызова функции.

Рассмотрим типичные ситуации, связанные с областью действия переменных, на следующих примерах.

Примеры

1. Переменная S определена только в теле функции, но продолжает существовать во внешней программе:

```
function Srectangle(width, height){
5 = width * height
return S
}
z = Srectangle(2, 3)    /* значения и z и S равны 6;
                           S - глобальная переменная */
```

2. Переменная 5 определена во внешней программе, но используется в теле функции:

```
function Srectangle(width, height) {
S = width * height
return 5
}
S = 2
z = Srectangle(2, 3)    /* значение z равно 6,
                           а значение 5 осталось равным 2,
                           то есть значению глобальной переменной;
                           определенной во внешней программе */
```

3. То же, что и в примере 2, но с использованием ключевого слова **van**

```
function Srectangle(width, height){
var 5 = width * height
return S
I
var S =2
z = Srectangle(2, 3)    /* значение z равно 6,
                           а значение S осталось равным 2,
                           то есть значению глобальной переменной,
                           определенной во внешней программе */
```

4. Исключительно локальные переменные:

```
function Srectangle(width, height){
var S = width * height
var x =17
return S
}
z = Srectangle(2,3)     /* значение z равно 6;
                           переменная S во внешней программе
                           не определена;
                           переменная x во внешней программе
                           не определена */
```

5. Имена параметров в определении функции совпадают с именами параметров во внешней программе и с параметрами в вызове функции:

```
function Srectangle(width, height) {
var s = width * height
width = width + 10              // изменяем значение параметра width
return s
I
width = 2
height = 3
z = Srectangle(width, height)   /* значение z равно 6;
                                   значение переменной width равно 2,
                                   то есть осталось без изменений */
```

Программа может содержать и определение функции, и выражения ее вызова. При этом порядок их следования в программе не важен: вы можете сначала написать определение функции, а затем где-нибудь в программе написать ее вызов, или,

наоборот, написать сначала вызов функции, а ее определение разместить где-нибудь в конце программы или даже в отдельном файле с расширением y>. В приведенных выше примерах мы записывали определение функции до ее вызова, однако программисты чаще поступают наоборот.

Пример

Функция вычисления факториала п, с помощью оператора цикла.

В предыдущем разделе, посвященном операторам цикла, мы рассматривали программный код вычисления факториала числа п. Напомним, что факториал целого положительного числа п в математике обозначают как п! и вычисляют по формуле: 1*2*3*...*п,если п<=1. Для п=0 и п=1 п!=1. В этом примере программу вычисления п! мы оформим в виде функции:

```
function  factorial(n){
if(n <= 1)  return 1
rezult  =2                        // result - переменная для результата
for (i = 3; i< = n; i++) {
   rezult = rezult*i
}
return  result
```

Для вычисления факториала конкретного числа, например 12, следует записать выражение **factorial(12).**

Если в нашей программе определена описанная выше функция **factorialQ,то** в этой же программе возможен такой фрагмент, содержащий вызов этой функции:

```
var m = 10
x = factorial(m))                // значение x равно 3628800
```

Определение функции может содержать вызов этой же функции — так называемое рекурсивное определение функции. В качестве примера рекурсивной функции в учебной литературе обычно приводят функцию вычисления факториала.

Пример

Функция вычисления факториала п! с помощью рекурсии.

```
function factorial(n) {
   if(n <= 1){
   return 1
   }
   return n*factorial(n-1)       // вызов функции factorial()
}
```

Отметим, что определение этой функции можно записать более компактно, опустив пару фигурных скобок, поскольку заключенный в них блок кода содержит лишь одно выражение:

```
function  factorial(n){
if(n <= 1)  return 1
return  n*factorial (n-1)        // вызов функции factorial O
}
```

СОВЕТ

> Чтобы избежать ошибок, старайтесь не использовать в коде функций имена переменных, инициализированных во внешней программе. По возможности стремитесь к тому, чтобы все, что делается внутри вашей функции, было локальным, формально независимым от внешнего окружения.

Определение функции может содержать в себе определения других функций, однако такие вложенные функции доступны только из кода функции, содержащей их определения.

Определение функции создает так называемый экземпляр объекта Function, обладающий полезными свойствами и методами. В частности, он имеет свойство arguments, содержащее информацию о параметрах, которые действительно были переданы функции при ее вызове. В некоторых случаях, прежде чем выполнить выражения тела функции, требуется проанализировать значения параметров. И тогда понадобится использовать свойство arguments. Подробности описаны ниже, в подразделе 1.7.7.

1.6.3. Выражения с функциями

Выше мы рассматривали, что такое выражения с операторами. Ко множеству элементарных выражений JavaScript можно добавить еще и вызовы функций. Тогда вызов функции может быть правой частью оператора присвоения, а также составной частью любого выражения с операторами.

Примеры

1. Пусть **Srectangle(width,** height) — функция, возвращающая значение площади прямоугольника со сторонами width и height. Тогда для вычисления площади прямоугольного треугольника с катетами a и b можно использовать следующее выражение:

   ```
   S3 = 0.5* Srectangle(a, b)
   ```

   В этом примере вызов функции является операндом выражения с арифметическим оператором умножения. При этом значение выражения присваивается переменной S3.

2. Вызов функции может использоваться в логических выражениях:

   ```
   if (Srectangle(a, b) > Srectangle(c, d))
   alert("Первый прямоугольник больше второго")
   ```

3. В качестве параметра функции может указываться вызов другой функции:

   ```
   var x = "25px"
   var y = 12
   var S = Srectangle(parseInt(x) , y)
   ```

В выражениях с вызовами функций последние имеют наивысший приоритет.

1.7. Встроенные объекты

Объекты представляют собой программные единицы, обладающие некоторыми свойствами. Об объекте мы можем судить по значениям его свойств и описанию того, как он функционирует. Программный код встроенных в JavaScript объектов нам недоступен. Главное для нас сейчас — усвоить, как пользоваться объектами. Это совсем просто, нужно только соблюдать нехитрый синтаксис. Использовать объекты не труднее, чем функции. Управление веб-страницами с помощью сценариев, написанных на JavaScript, заключается в использовании и изменении

свойств объектов HTML-документа и самого браузера. Эти вопросы мы отложим до следующих глав.

Встроенные объекты имеют фиксированные названия и свойства. Все свойства этих объектов разделяют на два вида: просто свойства и методы. Свойства аналогичны обычным переменным. Они имеют имена и значения. Некоторые свойства объектов доступны только для чтения. Это означает, что их значения нельзя изменять. Другие свойства доступны и для записи — их значения можно изменять с помощью оператора присвоения. Методы аналогичны функциям, они могут иметь параметры или не иметь их.

Чтобы узнать значение свойства объекта, необходимо указать имя этого объекта и имя свойства, отделив их друг от друга точкой: имя_объекта.свойство. Заметим, что объект может и не иметь свойств.

Мы можем заставить объект выполнить тот или иной присущий ему метод. В этом случае также говорят о применении метода к объекту. Синтаксис соответствующего выражения такой: имя_объекта.метод(параметры). Заметим, что объект может не иметь методов.

Итак, по синтаксису свойства отличаются от обычных переменных тем, что имеют составные имена, а также тем, что значения некоторых свойств нельзя изменить. Методы отличаются с точки зрения синтаксиса от обычных функций только тем, что имеют составные имена.

В свете изложенного выше объект можно понимать как некоторый контейнер, содержащий переменные-свойства и функции-методы. Разумеется, в этом контейнере есть еще что-то, но оно скрыто от нас. Мы можем воздействовать на объект только с помощью свойств и методов. В популярных книгах часто сравнивают объект с черным ящиком, у которого есть входы и выходы, доступные для наблюдения и, возможно, для управления. Я не буду развивать эту аналогию, поскольку для новичков, не обремененных традициями программирования без объектов, понять, что такое объекты и как ими пользоваться, не составляет особого труда. Для них понятия объекта и функции являются понятиями примерно одинаковой сложности. При этом сложность легко преодолима, было бы желание.

В JavaScript математические вычисления, сложная обработка строк и дат, а также создание массивов производятся с помощью соответствующих встроенных объектов. Для разработчиков веб-сайтов особенно важны объекты String (обработка строк), Array (массивы), Math (математические формулы и константы) и Date (работа с датами). Обратите на них особое внимание.

Встроенные объекты, как уже отмечалось, имеют фиксированные названия. Объекты с именами, совпадающими с их фиксированными названиями, называются статическими. Однако вы можете создать экземпляры (копии) статических объектов, присвоив им свои собственные имена. Экземпляры статических объектов являются объектами в вашей программе, которые наследуют от первых все их свойства и методы. Экземпляры объектов — это некоторые частные воплощения в программе соответствующих статических объектов. Вместе с тем вы можете использовать и статические объекты в чистом виде, не создавая никаких их копий. Например, для формульных вычислений используется статический объект Math, а в случае массивов создаются экземпляры статического объекта Array, содержа-

щие конкретные данные, к которым применимы все общие методы и свойства статического объекта Array.

Встроенные объекты имеют, среди прочих, свойство prototype (прототип), с помощью которого можно добавлять новые свойства и методы к уже существующим экземплярам объектов. Эти новые свойства и методы, разумеется, вы должны предварительно сами продумать и воплотить в виде программных кодов. Ниже мы покажем, как это делается. Например, при желании вы можете создать свой собственный метод обработки строк или массивов и присоединить их к конкретным объектам, чтобы затем использовать также, как и встроенные свойства и методы. При разработке сценариев для веб-страниц такая задача редко возникает, но JavaScript предназначен не только для создания сценариев.

1.7.1. Объект String (Строка)

Объект String представляет интерес главным образом благодаря методам обработки строк. Он незаменим, когда требуется, например, найти позицию вхождения одной строки в другую, вырезать из строки некоторую ее часть, разбить строку на отдельные элементы и создать из них массив и т. д.

С помощью объекта String можно создать строку как строковый объект. Однако в подавляющем большинстве случаев для этого достаточно использовать обычную переменную и оператор присвоения строкового значения. В этом случае интерпретатор все равно создает экземпляр (копию) строкового объекта, свойства и методы которого доступны из программного кода.

Создание строкового объекта

Для создания строкового объекта используется выражение следующего вида:

```
имя_переменной = new String("строковое_значение")
```

Здесь имя_переменной выполняет роль ссылки на строковый объект. Например, выражение mystring = new 5Ыпд("Привет!") создает строковый объект mystring со значением "Привет!".

Однако можно создать строковый объект и с помощью обычного оператора присвоения:

```
имя_переменной = "строковое_значение"
```

или

```
var имя_переменной = "строковое_значение"
```

Доступ к свойствам и методам строкового объекта обеспечивается такими выражениями:

```
строка.свойство
String.свойство

строка.метод([параметры])
String.метод([параметры])
```

Некоторые методы могут и не иметь параметров, что указано с помощью квадратных скобок. Здесь *строка* может быть ссылкой на строковый объект, строковой переменной, выражением, возвращающим строку, а также просто строковым значением.

Когда используется ключевое слово String в качестве имени объекта, это означает, что нас интересуют свойства и методы статического строкового объекта, то есть общие свойства и методы, не связанные, вообще говоря, с конкретными свойствами и методами конкретного строкового объекта (экземпляра объекта String).

Ниже приведены три различных способа использования свойства length строкового объекта, значением которого является длина строки (количество символов в строке).

```
mystring = "Однажды в студеную зимнюю пору"
mystring.length                              // значение равно 30
"Однажды в студеную зимнюю пору".length      // значение равно 30

function fstring(){return "abcde"}           /* функция, возвращающая
                                                строку "abcde" */
fstringO.length //значение равно 5
```

Методы строкового объекта используются для синтаксической обработки и форматирования строк. Эти две группы методов мы рассмотрим отдельно.

Свойства String

length — длина или, иными словами, количество символов (включая пробелы) в строке; целое число.

Пример

```
"Иван".length              // значение равно 4
"ПриветХпвсем". length     /* значение равно 11
                              (\п - один символ перевода строки) */
x = ""                     // пустая строка
x.length                   // значение равно 0 (пустая строка имеет длину 0)
```

prototype — свойство (прототип), позволяющее добавить новые свойства и методы ко всем создаваемым строковым объектам, если уже существующих вам окажется недостаточно.

Пример

В приведенном ниже примере мы создаем новый метод для всех строковых объектов. Содержание этого метода определяется пользовательской функцией.

```
function myFuncO () {          // функция для нового метода
return "Вадим"
I
                               // Добавление нового метода myName к прототипу:
String.prototype.myNameO = myFunc
mystring = "Автор этой книги - " + "Дунаев ".myNameO
                               /* значение mystring
                                  равно "Автор этой книги - Дунаев Вадим" */
```

Методы String обработки строк

1. charAt(nHfleKc) — возвращает символ, занимающий в строке указанную позицию.

   Синтаксис: строка.c1пагA1(индекс)

   Возвращает односимвольную или пустую строку.

   Параметр (индекс) является числом, индекс первого символа равен 0.

Примеры

```
"Привет".charAt(2)                       // значение равно "и"
"Привет".charAt(15)                      // значение равно ""
mystring = "Привет"
mystring.charAt(mystring.length-1)       /* значение последнего символа
                                            равно "т" */
```

2. charCodeAt([MHfleKc]) — преобразует символ в указанной позиции строки в его числовой эквивалент (код).

   Синтаксис: строка. сИагСойеАЩиндекс])

   Возвращает число.

   IE4+ и NN6 поддерживают систему кодов Unicode , NN4 — ISO-Latin 1.

   Примеры

```
"abc" . charCodeAtO                      // значение равно 97
"abc" . charCodeAt(l)                    // значение равно 98
"abc" . charCodeAt(25)                   // значение равно NaN
"" . charCodeAt (25)                     // значение равно NaN
"я". charCodeAt(0)                       // значение равно 1103
```

3. fromCharCode(HOMepl [, номер2 [, ... номерМ]]) — возвращает строку символов, числовые коды которой указаны в качестве параметров.

   Синтаксис: String.fromCharCode(HOMepl [, номер2 [,... номерМ]])

   Возвращает строку.

   IE4+ и NN6 поддерживают систему кодов Unicode, NN4 — ISO-Latin 1.

   Пример

```
String.fromCharCode(97,98,1102)          // значение равно "aЬю"
```

4. concat(crpoKa) — конкатенация (склейка) строк.

   Синтаксис: строка!.concal(строка2)

   Возвращает строку.

   Этот метод действует так же, как и оператор «+» сложения для строк: к строке строка! приписывается справа строка2.

   Примеры

```
"Иван".concat("Иванов")                  // значение равно "ИванИванов"
x = "Федор" + " "                        // значение x равно "Федор "
x.concat("Иванов")                       // значение равно "Федор Иванов"
```

5. indexOf(cTpoKa_noMCKa [, индекс]) — производит поиск строки, указанной параметром, и возвращает индекс ее первого вхождения.

   Синтаксис: строка.1пс1exOт(строка_поиска [, индекс])

   Возвращает число.

   Метод производит поиск позиции первого вхождения строка_поиска в строку строка. Возвращаемое число (индекс вхождения) отсчитывается от 0. Если поиск не удачен, то возвращается -1. Поиск в пустой строке всегда возвращает -1. Поиск пустой строки всегда возвращает 0.

   Второй параметр, не являющийся обязательным, указывает индекс, с которого следует начать поиск.

Этот метод хорошо использовать вместе с методом выделения подстроки substr() (см. ниже), когда требуется сначала определить позиции начала и конца выделяемой подстроки. Рассматриваемый здесь метод подходит для определения начальной позиции.

Примеры

```
x ="Во первых строках своего письма"
x.indexOf("первых")                  //значение равно 3
x.indexOf("первых строках")          //значение равно 3
x.indexOf("вторых строках")          //значение равно -1
x.indexOf("В")                       //значение равно 0
x.indexOf ("в")                      //значение равно 6
x.indexOf("в" , 7)                   //значение равно 19
x.indexOf(" ")                       //значение равно 2
x.indexOf(" ", 5)                    //значение равно 9
x.indexOf("")                        //значение равно 0
```

6. **LastIndexOf(cTpoKa_noHCKa [, индекс])** — производит поиск строки, указанной параметром, и возвращает индекс ее первого вхождения; при этом поиск начинается с конца исходной строки, но возвращаемый индекс отсчитывается от ее начала, то есть от 0.

   Синтаксис: строка.LastIndexOf(crpoKa_noMCKa [, индекс])

   Возвращает число. Метод аналогичен рассмотренному выше **indexOfQ** и отличается лишь направлением поиска.

   Примеры

```
x ="Во первых строках своего письма"
x.lastIndexOf("первых")              // значение равно 3
x.lastIndexOf("a")                   // значение равно 30
x.indexOf("a")                       // значение равно 15
```

7. **1oca1eCompare(строка)** — позволяет сравнивать строки в кодировке Unicode, то есть с учетом используемого браузером языка общения с пользователем.

   Синтаксис: **строка!.1oca1eCompare(строка2)**

   Возвращает число. Совместимость: IE5.5+, NN6+.

   Если сравниваемые строки одинаковы, метод возвращает 0. Если **строка!** меньше, чем строка2, то возвращается отрицательное число, в противном случае - положительное. Сравнение строк происходит путем сравнения сумм кодов их символов. Абсолютное значение возвращаемого числа зависит от браузера. Так, IE5.5 и IE6.0 возвращают 1 или -1, а NN6 — разность сумм кодов в кодировке Unicode.

8. **зИ$_c$е(индекс1 [,индекс2])** —возвращает подстроку исходной строки, начальный и конечный индексы которой указываются параметрами, за исключением последнего символа.

   Синтаксис: строка.$Hce(индекс1 [, индекс2])

   Возвращает строку. Данный метод не изменяет исходную строку.

   Если второй параметр не указан, то возвращается подстрока с начальной позицией **индекс!** и до конца строки. Отсчет позиций начинается с начала строки. Первый символ строки имеет индекс 0. Если второй параметр указан, то возвращается подстрока исходной строки начиная с позиции **индекс!** идо позиции

индекс2, исключая последний символ. Если второй параметр отрицателен, то отсчет конечного индекса производится от конца строки. В этом заключается основное отличие метода slice() от substr(). Сравните также этот метод с методом substringQ.

Примеры

```
x = "Во первых строках своего письма"
x.slice(3,8)            // значение равно "первы"
x.slice(3,-2)           // значение равно "первых строках своего пись"

/* Анализ адреса электронной почты */
x = "mumuisigerasim.ru"
i = x. indexOf ("@")                        // значение равно 4
_name = x.slice(0, i)                       // значение равно "mumu"
_domen = x.slice(i+l, x. indexOf (".")) )   // значение равно "gerasim"
_domen = x.slice(i+l, -3)                   // значение равно"gerasim"
```

9. зрЩразделитель [, ограничитель]) — возвращает массив элементов, полученных из исходной строки.

Синтаксис: строка.зрЩразделитель [, ограничитель])

Возвращает массив.

Первый параметр является строкой символов, используемой в качестве разделителя строки на элементы. Второй необязательный параметр — число, указывающее, сколько элементов строки, полученной при разделелении, следует включить в возвращаемый массив.

Если разделитель — пустая строка, то возвращается массив символов строки.

Примеры

```
x = "Привет всем"
x.split(" ")    /* значение - массив из элементов: "Привет", "всем" */
x.split("е")    /* значение - массив из элементов: "Прив", "т вс", "м" */
x.split("е",2)  /* значение - массив из элементов: "Прив", "т вс" */
```

10. substr(nHfleKC [, длина]) — возвращает подстроку исходной строки, начальный индекс и длина которой указываются параметрами.

Синтаксис: строка.substr(nHfleKc [, длина])

Возвращает строку. Данный метод не изменяет исходную строку.

Если второй параметр не указан, то возвращается подстрока с начальной позицией индекс! и до конца строки. Отсчет позиций начинается с начала строки. Первый символ строки имеет индекс 0. Если второй параметр указан, то возвращается подстрока исходной строки начиная с позиции индекс 1 и с общим количеством символов, равным длина. Сравните этот метод с методами sliceQ и substringQ.

Примеры

```
x = "Привет всем"
x.substr(7,4)                   // значение равно "всем"

/* Анализ адреса электронной почты */
x = "mumu@gerasim.ru"
i = x.indexOf("@")              // значение равно 4
_name = x.substr(0, i)          // значение равно "mumu"
_domen = x.substr(i+1)          // значение равно "gerasim.ru"
```

11. substring(MHfleKd, индекс2) — возвращает подстроку исходной строки, начальный и конечный индексы которой указываются параметрами.

 Синтаксис: CTpoKa.substring(MHfleKCl, индекс2)

 Возвращает строку. Данный метод не изменяет исходную строку.

 Порядок индексов не важен: наименьший из них считается начальным. Отсчет позиций начинается с начала строки. Первый символ строки имеет индекс 0. Символ, соответствующий конечному индексу, не включается в возвращаемую строку. Сравните этот метод с методами substrQ и sliceQ.

 Примеры

```
x = "Привет всем"
x.substring(0^, 6)              // значение равно "Привет"
x. substring(7, x. length)      // значение равно "всем"
x. substring(7, 250)            // значение равно "всем"
x.substring(250, 7)             // значение равно "всем"
```

12. toLocaleLowerCaseQ,toLowerCase() —переводят строку в нижний регистр.

 Синтаксис: cTpOKa.toLocaleLowerCaseQ, CTpoKa.toLowerCaseQ

 Возвращают строку. Первый метод работает в IE5.5+, NN6, учитывает определенные языковые системы.

 Приведение строк к одному и тому же регистру требуется, например, при сравнении содержимого строк без учета регистра. Кроме того, многие серверы чувствительны к регистру, в котором определены имена файлов и папки (обычно требуется, чтобы они были определены в нижнем регистре).

 Примеры

```
x = "ЗдраВствуйТе"
x.toLocaleLowerCase()            // значение равно "здравствуйте"
x. toLowerCase()                 // значение равно "здравствуйте"
y = "Здравствуйте"
x == y                           // значение равно false
x. tol_owerCase() == y . toLowerCase    // значение равно true
```

13. toLocalellpperCaseQ, toUpperCase() — переводят строку в верхний регистр.

 Синтаксис: CTpoKa.toLocaleUpperCaseQ, crpoKa.toUpperCase()

 Возвращают строку.

 Первый метод работает в IE5.5+, NN6, учитывает определенные языковые системы.

 Приведение строк к одному и тому же регистру требуется, например, при сравнении содержимого строк без учета регистра. Кроме того, многие серверы чувствительны к регистру, в котором определены имена файлов и папки.

 Примеры

```
x = "ЗдраВствуйТе"
x. toLocaleUpperCase()           // значение равно "ЗДРАВСТВУЙТЕ"
x.toUpperCase()                  // значение равно "ЗДРАВСТВУЙТЕ"
y = "Здравствуйте"
x == y                           // значение равно false
x . toUpperCaseO == y . tollpperCase    // значение равно true
```

Методы String форматирования строк

Как известно, тексты на веб-страницах обычно создаются и форматируются с помощью тегов HTML. Однако тексты можно создавать на веб-страницах и с помощьюсценариев. Например, чтобывывестинавеб-страницу строку "Приветвсем!" полужирным шрифтом, в HTML-коде следует написать следующую инструкцию:

```
<B>Привет всем!</B>
```

Чтобы подготовить эту же строку в таком же формате средствами JavaScript, в сценарии следует написать такое выражение:

```
"Приветвсем!".bold()
```

Здесь использован метод boldQ строкового объекта для форматирования строк. Выполнение этого выражения лишь создает отформатированную строку, но не выводит ее в окно браузера. Чтобы сделать это, следует еще выполнить метод write() объекта document для записи этой строки в HTML-документ. Ниже приведен пример кода и вид HTML-документа в окне браузера (рис. 1.11).

```
<HTML>
<SCRIPT>
x = "Привет всем!".bold()
document.wri te(x)
</SCRIPT>
<P>Приветствие было вставлено сценарием JavaScript
</HTML>
```

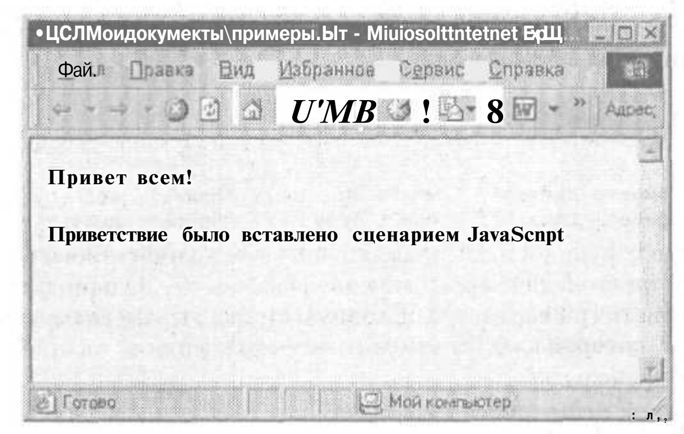


Рис. 1.11. Приветствие в верхней строке сформировано сценарием

В рассмотренном выше примере использован объект document и его метод writeQ. Другие свойства и методы этого объекта рассматриваются в следующих главах.

Методы форматирования строк носят названия, соответствующие тегам HTML. Их особенность в том, что, в отличие от тегов, их следует записывать только в нижнем регистре (строчными буквами). Синтаксис такой же, как и у ранее рассмотренных методов:

```
строка.метод(параметр)
```

Большинство методов форматирования не имеют параметров. Далее приведен их список.

anchor("anchor_имя")
blink()
bold()
fixed()
fontcolor(3Ha4eHne_L|BeTa)
fontsize(4MOio от 1 до 7)
italicsQ

Ип1<(расположение или URL)
ig()
smalt()
strike()
sub()
sup()

Примеры

```
"Глава 2" . anchor("volume2"))        /* эквивалентно HTML-коду:
                                       <A NAME="volume2">rnaea 2</A> */
"Страница автора".link("http:/www.admiral.ru/~dunaev")
     /* эквивалентно HTML-коду:
        <A HREF = "http:/www.admiral.ru/~dunaev")>Страница автора</A> */
```

Функции вставки и замены подстрок

При обработке строк часто требуется вставить или заменить подстроки. Удаление подстроки из данной строки можно рассматривать как частный случай замены, а именно замены указанной подстроки пустой строкой. С помощью методов объекта String можно написать программу для решения этой задачи. Поскольку она часто встречается, то целесообразно оформить программу в виде функций.

Рассмотрим сначала функцию вставки строки в исходную строку. Назовем ее, например, insstr. Данная функция должна принимать три параметра: исходную строку si, вставляемую строку s2 и индекс позиции вставки п. Ниже приведены ее определение и примеры вызова:

```
function insstr(sl,s2,n) {
return si.slice(O.n) + s2 + sl.slice(n)
}

insstr ("Привет., друзья", " мои", 7)       // "Привет, мои друзья"
insstr("Привет, друзья", " мои", 100)    // "Привет, друзья мои"
```

Теперь займемся функцией, которая заменяет в исходной строке все вхождения заданной подстроки на подстроку замены. Назовем эту функцию replacestr. Она должна принимать три параметра: исходную строку si, заменяемую подстроку s2 и подстроку s3, которой следует заменить все вхождения s2 в si (s2 может встречаться в si несколько раз).

Очевидно, прежде всего необходимо найти все вхождения s2 в si. Если исходная строка не содержит в себе подстрок s2, то функция должна вернуть исходную подстроку без всяких изменений. В противном случае требуется изъять из s1 все вхождения s2, а на их место вставить подстроку s3. Полученная таким образом строка должна возвращаться функцией в качестве результата. Ниже приведено определение функции для замены подстрок и два примера ее вызова:

```
function replacestr(si, s2, s3) {
  var s = "                               // обработанная часть строки
  while (true) {
    i = si.indexOf(s2)2)                  // индекс вхождения s2 в si
    if (i >= 0) {
    s = s + sl.substr(0, i) + s3          // обработанная часть строки
    s1 = s1.substr(1 + s2.length)         // оставшаяся часть строки
    }else break
```

```
    }
    return s + sx
\

replacestr C'bacdae", "a", "X")          //  "bXcdXe"

x = "Иван Иванов"
replacestr(x,"Иван","Федор")             // "Федор Федоров"
```

Заметим, что если заменяемая подстрока s2 является пустой, то цикл будет продолжаться бесконечно, поскольку пустая строка входит в состав любой строки. Чтобы избежать зацикливания, следует немного изменить код тела функции. Но что должна возвращать наша функция, если s2 — пустая строка? Вы можете решить, что в этом случае следует просто вернуть исходную строку si или же, например, считать пустую строку строкой, содержащей пробел. Последний вариант представляется мне более интересным. Вот код, реализующий эту идею:

```
function replacestr(si, s2, s3) {
   if (s2 == "")
   s2 = " "                  // заменяем пустую строку на строку с пробелом
   var s = "                 // обработанная часть строки
   while (true) {
      i = si. indexOf (s2)   // индекс вхождения s2 в si
      if (i >= 0) {
         s = s + sl.substr(0, i) + s3      // обработанная часть строки
         si = sl.substr(i + s2.length)     // оставшаяся часть строки
      }else break                          // выход из цикла
   }
   return s + sx
}
```

Рассмотренные выше функции вставки и замены подстрок (insstrQ и replacestrQ) готовы к практическому применению в программах, где требуется обработка строк. Вы можете сохранить определения этих функций в текстовом файле с расширением .js, который будет играть роль библиотеки функций. Этот файл можно вызывать из основной программы, когда он потребуется. Более подробно этот вопрос мы рассмотрим в следующей главе.

Функции удаления передних и заключительных пробелов

При обработке строк (например, введенных пользователем в поля формы) нередко возникает задача удалить лишние передние и задние пробелы. Многие языки программирования имеют соответствующие встроенные функции, но в JavaScript их нет. Поэтому не помешает создать их самому с помощью имеющихся средств и поместить в свою библиотеку (сохранить в текстовом файле с расширением .js).

Решить поставленную задачу можно несколькими способами. Я придумал следующий алгоритм. Преобразуем исходную строку в массив слов, используя в качестве разделителя один пробел " ". Затем необходимо проанализировать первые или последние элементы массива в зависимости от того, что нам требуется: удалить передние или задние пробелы в исходной строке. Допустим, необходимо удалить передние пробелы. Тогда, если исходная строка содержала N пробелов, первые N элементов массива будут содержать пустые строки"". Мы проверяем в цикле значения первых элементов, пока не найдем непустой элемент или пока не исчер-

паем весь массив. Если первый непустой элемент массива имеет индекс i, то создадим новый массив, содержащий элементы исходного, начиная с этого индекса. Наконец, склеим элементы этого массива в одну строку, используя в качестве разделителя строку с единственным пробелом " ". Все, что нам понадобится, — это методы объекта String, оператор цикла и оператор условного перехода.

Сначала рассмотрим функцию ItrimQ, удаляющую передние пробелы из исходной строки.

```
function ltrim(xstr){
if (!(xstr.indexOf(" ") == 0))
   return xstr                      /* вернуть исходную строку,
                                       если в ней нет передних пробелов */
var astr = xstr. split(" ")         // создаем массив из слов строки
var i = 0
while (i < astr.length){
   if (!(astr[i] == ("")))
      break                         /* выходим из цикла,
                                       если элемент не пуст */
   i++
}
astr = astr.slice(i)                // создаем массив
return astr.join("")                // склеиваем элементы массива в строку
}
```

Функция rtrim() для удаления заключительных пробелов в строке устроена аналогично. Ее отличие в том, что поиск пробелов происходит с конца строки. Обратите внимание, что счетчик циклов в этом случае убывающий.

```
function rtrim(xstr){
if (! (xstr.lastIndexOf(" ") == xstr.length - 1))
   return xstr
var astr = xstr.split(" ")
var i = astr.length -1
while (i>0){
      if (!(astr[i] == ("")))
         break
      i --
}
astr = astr.slice(0, i+1)
return astr.join(" ")
}
```

Чтобы удалить из строки и передние, и концевые пробелы, достаточно выполнить следующее выражение:

```
ystr = rtrim(ltrim(xstr))
```

Впрочем, можно создать специальную функцию trim(), которая тримингует строку:

```
function trim(xstr) {
   return rtrim(ltrim(xstr))

:)
```

1.7.2. Объект Array (Массив)

Массив представляет собой упорядоченный набор данных. Его удобно представить себе в виде одностолбцовой таблицы, содержащей некоторое количество строк. В ячейках такой таблицы могут находиться данные любого типа, в том чи-

еле и массивы. В последнем случае можно говорить о многомерных массивах (то есть о массивах массивов). Количество элементов в массиве (строк в таблице) называется длиной массива. К элементам массива можно обращаться в программе по их порядковому номеру (индексу). Нумерация элементов массива начинается с нуля, так что первый элемент имеет индекс 0, а последний — на единицу меньший, чем длина массива.

Массивы применяются во многих более или менее сложных программах обработки данных, а в некоторых случаях без них просто не обойтись. Если среди используемых данных есть группы таких, которые обрабатываются одинаковым образом, то, возможно, лучше организовать их в виде массива.

Создание массива

Существует несколько способов создания массива. В любом случае прежде всего создается новый объект массива с использованием ключевого слова new:

```
имя_массива = new Array([длина_массива])
```

Здесь **длина_массива** является необязательным числовым параметром, о чем говорят квадратные скобки. Если длина массива не указана, то создается пустой массив, не содержащий ни одного элемента. В противном случае создается массив с указанным количеством элементов, однако все они имеют значение null (то есть не имеют значений).

Вы можете сначала создать пустой массив, а затем добавить к нему нужное количество элементов с помощью оператора присвоения. Заметим, что выражение с ключевыми словами new Array создает экземпляр (копию) объекта Array.

У объекта Array имеется свойство length, значением которого является длина массива. Чтобы получить значение этого свойства, необходимо использовать выражение **имя_массива.length.**

Создав массив, можно присвоить значения его элементам, используя для этого оператор присвоения. В левой части оператора присвоения указывается имя массива, а рядом с ним в квадратных скобках индекс элемента. Мы уже говорили, что к элементам массива обращаются по индексу: **имя_массива[индекс].** Здесь квадратные скобки обязательны.

Рассмотрим создание массива earth, содержащего в качестве элементов некоторые характеристики нашей планеты. Обратите внимание, что элементы в этом массиве различных типов (строковые и числовые).

```
earth = new Array(4)          // массив из 4-х элементов
earth[0] = "Планета"
earth fl] = "24 часа"
earth[2] = 6378
earth[3] = 365.25
earth.length                  // значение равно 4
```

Если нам потребуется значение, например, третьего элемента массива, то достаточно использовать выражение earth **[2]**.

Другой способ создания массива заключается в непосредственном определении элементов в круглых скобках за ключевым словом Array.

Пример

```
earth = new Array("Планета", "24 часа", 6378, 365.25)
```

JavaScript автоматически создает индексы для элементов массива, так что к элементам массива, созданного таким способом, также можно обращаться по индексам.

Третий способ создания массива — присвоить имя каждому элементу, подобно имени свойства объекта

Пример

```
earth = newArrayO                // пустой массив
earth. xtype = "Планета"
earth.xday = "24 часа"
earth.radius = 6378
earth.period = 365.25
```

В этом случае обращение к элементу происходит как к свойству объекта, например **earth.radius.** Возможен и такой вариант: **earth["radius"].** Однако по индексу к элементам в таком массиве обращаться нельзя.

Многомерные массивы

Массивы, рассмотренные выше, являются одномерными. Их можно представить себе в виде таблицы из одного столбца. Однако элементы массива могут содержать данные различных типов, в том числе и объекты, а значит, и массивы. Если в качестве элементов некоторого одномерного массива создать массивы, то получится двухмерный массив. Обращение к элементам такого массива происходит в соответствии со следующим синтаксисом:

```
имя_массива[индекс_уровня1] [индекс_уровня2]
```

Если массив имеет размерность, большую двух, то синтаксис обращения к массивам имеет аналогичный синтаксис: следует добавить нужное количество квадратных скобок, заключающих нужные индексы.

Типичным примером двухмерного массива является массив опций меню. У такого меню есть горизонтальная панель с опциями, называемая главным меню. Некоторым опциям главного меню соответствуют раскрывающиеся вертикальные подменю со своими опциями. Мы создаем массив, длина которого равна количеству опций главного меню. Элементы этого массива определяем как массивы названий опций соответствующих подменю. Чтобы была ясна структура нашей конструкции, мы выбрали названия опций надлежащим образом. Например, "Меню 2.1" — название 1-й опции подменю, соответствующего 2-й опции главного меню.

```
menu = new ArrayO
menu[0] = new anray("Мени) 1.1", "Меню 1.2", ". ","Меню 1.3")
menu[l] = new arrayC'Меню 2.1", "Меню 2.2")
menu[2] = new arrayC'Меню 3.1", "Меню 3.2" , "Меню 3.3", "Меню 3.4")
```

Чтобы обратиться ко 2-й опции 3-го подменю, следует написать:

```
menu[2][l]                              // значение равно "Меню 3.2"
```

Усложним нашу конструкцию, чтобы она содержала не только названия опций подменю, но и названия опций главного меню:

```
menu = new ArrayO
/* Массив опций главного меню: */
menu[0] = new Array("Меню!", "Меню2", "Меню3")
menu[l] = new ArrayO
menu[1][0] = new ArrayC'Меню 1.1". "Меню 1.2", "Меню 1.3")
menu[l] [l] = new ArrayC'Меню 2.1", "Меню 2.2")
menu[l][2] = new ArrayC'Меню 3.1", "Меню 3.2" , "Меню3.3", "Меню 3.4")
```

```
menu[0][l]                     // значение равно "Меню 2"
menu[0][2]                     // значение равно "Меню 3"
menu[l] [1] [0]                // значение равно "Меню 2. 1"
menu[1][2][3]                  // значение равно "Меню 3. 2"
```

Копирование массива

Иногда требуется создать копию массива, чтобы сохранить исходные данные и предохранить их от последующих модификаций. Например, метод сортировки элементов массива, который рассмотрен ниже, изменяет исходный массив. Однако для копирования массива недостаточно присвоить его другой переменной. Используя новую переменную и оператор присвоения, мы создаем лишь новую ссылку на прежний массив, а не новый массив.

Пример

```
a = new Array(5, 2, 4, 3)
x = a                  // ссылка на массив a
a[2] = 25              // изменение значения элемента с индексом 2
x[2]                   // значение равно 25, то есть новому значению
a [2]
```

В этом примере массивы a и x совпадают.

Чтобы скопировать массив, то есть создать новый массив, элементы которого равны соответствующим элементам исходного, следует воспользоваться оператором цикла, в котором элементам нового массива присваиваются значения элементов исходного, например:

```
a = new Array(5, 2, 4, 3)
x = new ArrayO                     // ссылка на массив a
for(i=0; i<a.length; i++)          /* копирование значений массива a
                                      в элементы массива x */
{x[i] = a [i] }
```

Свойства Array

1. length — длина или, иными словами, количество элементов в массиве; целое число.

   Синтаксис: имя_массива.length

   Поскольку индексация элементов массива начинается с нуля, индекс последнего элемента на единицу меньше длины массива. Это обстоятельство удобно использовать при добавлении к массиву нового элемента: myarray [mayarray.length] = значение.

2. prototype — свойство (прототип), позволяющее добавить новые свойства и методы ко всем созданным массивам.

   Например, следующее выражение добавляет свойство author ко всем уже созданным массивам:

   ```
   Array.prototype.author = "Иванов"
   ```

   Если теперь вместо Array, prototype написать имя существующего массива, то можно изменить значение свойства author только для этого массива:

   ```
   myarray = new ArrayO                  // создание массива myarray
   xarray = new ArrayO                   // создание массива xarray
   Array.prototype.author = "Иванов"     /* добавление прототипа
                                            ко всем массивам */
   ```

```
myarray.author = "Иванов младший"     /* изменение свойства author i
                                         для myarray */
xarray.author = "Сидоров"             /* изменение свойства author
                                         для xarray */
```

Прототипу можно присвоить функции. При этом они пополнят множество методов объекта Array.

Пример

Вычисление суммы элементов массива. Мы определяем функцию aSum(), которая возвращает сумму элементов числового массива. В качестве параметра эта функция принимает массив. Затем создаем конкретный массив чисел. Наконец, присоединяем к прототипу массива новый метод Sum — определенную ранее функцию aSum():

```
function aSum(xarray) {
   var s = 0
   for(i = 0; i <= xarray.length - 1; i++){
      s = s + xarray[i]
   }
   return s
I
myarray = new Array(2, 3,4)       // создаем массив myarray из 3-х чисел
Array.prototype.Sum = aSum        /* присоединяем метод (около имени
                                     функции скобки указывать не нужно) */
myarray.Sum(myarray)              /* применяем метод Sum к массиву myarray,
                                     передавая его в качестве параметра */
```

Этот пример призван просто проиллюстрировать использование свойства prototype. Чтобы вычислить сумму элементов массива, достаточно написать следующее выражение:

```
s = aSum(myarray)
```

Методы Array

Методы объекта Array предназначены для управления данными, сохраненными в структуре массива.

1. **сопса1(массив)** — конкатенация массивов, объединяет два массива в третий массив.

   Синтаксис: имя_массива!.сопса1(массив2)

   Возвращает массив. Данный метод не изменяет исходные массивы.

   Пример

   ```
   al = new array(l, 2, "Звезда")
   a2 = new arrayC'a", "б", "в", "г")
   a3 = al.concat(a3)             /* результат - массив с элементами:
                                     1, 2, "Звезда", "а", "б", "В", "Г" */
   ```

2. **p1'п(разделитель)** — создает строку из элементов массива с указанным разделителем между ними; является строкой символов (возможно, пустой).

   Синтаксис: **имя_массиваоо1'п(строка)**

   Возвращает строку символов.

   Примеры

   ```
   a = new array(l, 2, "Звезда")
   a.join(",")                    // значение - строка "1,2,Звезда"
   ```

```
a = new arrayd, 2, "Звезда")
a.join(" ")                          // значение - строка "1 2 Звезда"
```

3. pop() — удаляет последний элемент массива и возвращает его значение.

   Синтаксис: **имя_массива.pop()**

   Возвращает значение удаленного элемента массива. Совместимость: IE5.5+.

   Данный метод изменяет исходный массив.

4. ри5И(значение|объект) — добавляет к массиву указанное значение в качестве последнего элемента и возвращает новую длину массива.

   Синтаксис: имя_массива!.ри5Ь(значение|объект)

   Возвращает число. Совместимость: IE5.5+. Данный метод изменяет исходный массив.

5. **shiftQ** — удаляет первый элемент массива и возвращает его значение.

   Синтаксис: имя_массива.shiftQ

   Возвращает значение удаленного элемента массива. Совместимость: IE5.5+, NN4+. Данный метод изменяет исходный массив.

6. ип5И1'г1(значение|объект) — добавляет к массиву указанное значение в качестве первого элемента.

   Синтаксис: имя_массива.ип5МЩзначение|объект)

   Возвращает: ничего. Совместимость: IE5.5+. Данный метод изменяет исходный массив.

7. **reverseQ** — изменяет порядок следования элементов массива на противоположный.

   Синтаксис: имя_массива.геуегзе()

   Возвращает массив. Данный метод изменяет исходный массив.

   Пример

```
a = new array(I. 2, "Звезда")
a.reverse()              /* массив с элементами в следующем порядке:
                            "Звезда", 2, 1 */
```

8. 5и$е(индекс1 [, **индекс2])** — создает массив из элементов исходного массива с индексами указанного диапазона.

   Синтаксис: имя_массива.5Н5е(индекс! [, индекс2])

   Возвращает массив. Данный метод не изменяет исходный массив.

   Второй параметр (конечный индекс) не является обязательным, о чем свидетельствуют квадратные скобки в описании синтаксиса. Если он не указан, то создаваемый массив содержит элементы исходного массива начиная с индекса **индекс!** идо конца. В противном случае создаваемый массив содержит элементы исходного массива начиная с индекса индекс! идо индекса индекс2, за исключением последнего. При этом исходный массив остается без изменений.

   Пример

```
a = new array(1, 2, "Звезда","а", "Ь")
a.slise(1,3)             // массив с элементами: 2, "Звезда"
a.slise(2)               // массив с элементами: "Звезда", "а", "Ь"
```

9. 50|1([функция_сортировки]) — сортирует (упорядочивает) элементы массива с помощью функции сравнения.

   Синтаксис: имя_массива.5оИ:([функция_сравнения])

   Возвращает массив. Данный метод изменяет исходный массив. Параметр не обязателен, о чем свидетельствуют квадратные скобки.

   Если параметр не указан, то сортировка производится на основе сравнения ASCII-кодов символов значений. Это удобно для сравнения символьных строк, но не совсем подходит для сравнения чисел. Так, число 357 при сортировке считается меньшим, чем 85, поскольку сначала сравниваются первые символы и только в случае их равенства сравниваются следующие, и т. д. Таким образом, метод sortQ без параметра подходит для простой сортировки массива со строковыми элементами.

   Можно создать свою собственную функцию для сравнения элементов массива, с помощью которой метод sortQ отсортирует весь массив. Имя этой функции (без кавычек и круглых скобок) передается методу в качестве параметра. При работе метода функции передаются два элемента массива, а ее код возвращает методу значение, указывающее, какой из элементов должен следовать за другим. Допустим, сравниваются два элемента, x и y. Тогда в зависимости от числового значения (отрицательного, 0 или положительного), возвращаемого функцией сравнения, методом sort() принимается одно из трех возможных решений:

Значение, возвращаемое функцией сравнения	**Результат сравнения x и y**
<0	y следует за x
O	Порядок следования x и y не изменяется
>0	x следует за y

   Итак, по какому критерию сортировать элементы массива, определяется кодом функции сравнения. Если элемент массива имеет значение null, то в Internet Explorer он размещается в начале массива.

   Пример

```
myarray = new Array(4, 2, 15, 3, 30 )     // числовой массив
function comp(x, y) {          // функция сравнения return x-y
}
myarray.sort(comp)             /* массив с элементами в порядке:
                                  2, 3, 4. 15, 30 */
```

10. зрИсе(индекс, количество [, элем! [, элем2 [, ...э/ieMN]]]) — удаляет из массива несколько элементов и возвращает массив из удаленных элементов или заменяет значения элементов.

    Синтаксис: имя_массива.5рНсе(индекс, количество [, элем! [, элем2 [, ...элемM]]])

    Возвращает массив. Совместимость: IE5.5+. Данный метод изменяет исходный массив.

    Первые два параметра обязательны, а следующие — нет. Первый параметр является индексом первого удаляемого элемента, а второй — количеством удаляемых элементов.

Пример

```
a = new Array("Вася", "Иаван", "Марья", 12, 5)
x = a.splice(1,3)          /* x - массив элементов: "Иван", "Марья", 12
                              a - массив элементов: "Вася", 5 */
```

Метод **spliceQ** позволяет также заменить значения элементов исходного массива, если указаны третий и, возможно, последующие параметры. Эти параметры представляют значения, которыми следует заменить исходные значения элементов массива. При таком использовании метода spliceQ важен первый параметр (индекс), а второй (количество) может быть равным нулю. В любом случае, если количество элементов замены больше значения второго параметра, то часть элементов исходного массива будет заменена, а часть элементов будет просто вставлена в него. При этом метод spliceQ возвращает другой массив, состоящий из элементов исходного, индексы которых соответствуют первому и второму параметрам. Но это справедливо, если второй параметр не равен 0.

Пример

```
a = new Array("Вася", "Иван", "Марья", 12, 5)
x = a.splice(1,3, "Петр", "Кузьма", "Анна")
                 // x - массив элементов: "Иван", "Марья", 12
                 //a - массив элементов: "Вася","Петр", Кузьма", "Анна",5

a = new Array("Вася", "Иван", "Марья", 12, 5)
x = a.splice(1,0, "Петр", "Кузьма", "Анна", "Федор", "Ханс")
                 // x - пустой массив
                 /* a - массив элементов:
                   "Вася", "Петр", "Кузьма", "Анна", "Федор", "Ханс" */
```

H.toLocaleStringO, toString() — преобразуют содержимое массива в символьную строку.

Метод **toLocaleStringQ** поддерживается браузерами IE5.5+ и NN3+, а метод toStringO — и более ранними версиями. Алгоритм преобразования по методу **toLocaleStringO** зависит от версии браузера.

Для преобразования содержимого массива в строку рекомендую использовать методjoin().

Функции обработки числовых массивов

Во многих приложениях требуется получить статистические характеристики числовых данных, хранящихся в виде массива: сумму всех чисел, среднее, максимальное и минимальное значения. Здесь мы приведем коды функций, вычисляющих эти величины.

Функция, возвращающая сумму значений всех элементов непустого массива:

```
function S(aN) {
var S=aN[0]
for(var i = 1; i<= aN.length-1; i++){
          S += aN[i]
}
return S
}
```

Очевидно, для вычисления среднего значения следует просто воспользоваться выражением **S(aN)/aN.length.**

Функция, возвращающая минимальное значение среди элементов массива:

```
function Nmin(aN){
var Nmin = aN[0]
for(var i = l ; i <= aN.length-1; i++){
   if (aN[i] < Nmin)
      Nmin = aN[i]
I
return Nmin
}
```

Функция, возвращающая максимальное значение среди элементов массива:

```
function Nmax(aN){
var Nmax = aN[0]
for(var i = 1; 1 <= aN.length -1; i++){
   if (aN[i] > Nmax)
      Nmax = aN[i]
}
return Nmax
}
```

Мы можем создать одну функцию, которая вычисляет все перечисленные выше статистические характеристики и возвращает их как значения массива:

```
function statistic(aN) {
if (aN ==0 || aN == null || aN == "")
   return new Array(0,0,0,0)
var S = aN[0]
var Nmin = aN[0]
var Nmax = aN[0]
for(var i=l; i<=aN.length-1; i++){
   S += aN[i]
   if (aN[i] < Nmin)
      Nmin = aN[i]
   if (aN[i] > Nmax)
      Nmax = aN[i]
}
return new Array(S, S/aN.length, Nmin, Nmax)
}
```

В начале кода функции **statisticQ** мы проверяем, не является ли параметр пустым. Если это так, то все статистические характеристики считаются равными нулю.

1.7.3. Объект Number (Число)

При разработке веб-страниц математические операции используются не столь часто, в отличие от строковых. Обычно они связаны с изменением координат элементов страницы (свойства top, left, width, height таблицы стилей). Однако встречаются и более сложные случаи. Например, может потребоваться вычислить статистические характеристики данных, содержащихся в некоторой таблице. Так или иначе, в этих задачах необходимо иметь дело с числами. О числах мы уже говорили в разделе, посвященном типам данных. Теперь рассмотрим их более подробно.

Числа в JavaScript

В JavaScript числа могут быть только двух типов: целые и с плавающей точкой. Целые числа не имеют дробной части и не содержат разделительной точки. Числа с плавающей точкой имеют целую и дробную части, разделенные точкой.

Операции с целыми числами процессор компьютера выполняет значительно быстрее, чем операции с числами, имеющими точку. Это обстоятельство имеет смысл учитывать, когда расчетов много. Например, индексы, длины строкявляютей целочисленными. Число л, многие числа, полученные с помощью оператора деления, денежные суммы и т. п. являются числами с плавающей точкой.

В JavaScript можно производить операции с числами различных типов. Это очень удобно. Однако при этом следует знать, какого числового типа будет результат. Если результат операции является числом с дробной частью, то он представляется как число с плавающей точкой. Если результат оказался без дробной **части,** то он приводится к целочисленному типу, а не представляется числом, у которого в дробной части одни нули.

Примеры

```
2 + 3          // 7 - целое число
2 + 3.6        // 5.6 - число с плавающей точкой
2.4 + 3.6      // 6 - целое число
6.00           // число с плавающей точкой
```

Числа можно представлять и в так называемой экспоненциальной форме, то есть в формате: **число!ечисло2** или **число!Ечисло2.** Такая запись числа означает *число* ***110*ua"*** слo2.

Пример

```
1e5            // 100 000
2e6            // 2 000 000
1.5e3          // 1500
+1.5e3         // 1500
-1.5e3         // -1500
3e-4           // 0.ппоз
```

Числа в JavaScript можно представлять в различных системах счисления, то есть в системах с различными основаниями: 10 (десятеричной), 16 (шестнадцатеричной) и 8 (восьмеричной). К десятеричной форме представления чисел мы привыкли, однако следует помнить, что числа в этой форме не должны начинаться с О, потому что так записываются числа в восьмеричной системе.

Запись числа в шестнадцатеричной форме начинается с префикса Ох (или ОХ), где первый символ ноль, а не буква О, затем следуют символы шестнадцатеричных цифр: О, 1, 2,..., 9, а, Ь, с, d, е, f (буквы могут быть в любом регистре). Например, шестнадцатеричное число Ox4af в десятеричном представлении есть 1199.

Запись числа в восьмеричной форме начинается с нуля, за которым следуют цифры от 0 до 7. Например, 027 (в десятеричном представлении — 23). В арифметических выражениях числа могут быть представлены в любой из перечисленных выше систем счисления, однако результат всегда приводится к десятеричной форме.

Пример

Функция преобразования из десятеричной в шестнадцатеричную форму. Функция **to!6()** в этом примере принимает в качестве параметра десятеричное число и преобразует его в строку, содержащую это же число, но в шестнадцатеричной форме. При этом мы ограничиваемся числами, не превышающими 255 (в этом случае для представления числа потребуется не более двух шестнадцатеричных цифр).

```
function tol6(nlO) {
   hchars = "0123456789abcdef"   / строка, содержащая все 16-е цифры
   if (n!0 > 255) return null
```

```
    var i = n!0%16
    var j = (nl0 - i) /16
    rezult = "0x"
    rezult += hchars.charAt(j )
    rezult += hchars . charAt(i)
    return rezult
}

to!6(250)                    // "0xfa"
to!6(30)                     // "0xle"
to16(30)+10                 // "0xlel0" - склейка, а не сложение
parseInt(to16(30)) + 10      // 40
```

Напомним, что выражение вида x += у эквивалентно выражению x = x + у.

Функцию to!6() можно также создать и на основе массивов:

```
function to!6(n!0) {
    hchars = new Array ("0", "1 , "2", "3", "4", "5", "6", "2", "3", "4",
"a", "b", "c", "d", "e", "f"}
    if (n!0 > 255) return null
    var i = n!0%16
    var j = (n!0 - i)/16
    rezult = "0x"
    rezult += hchars [j]
    rezult += hchars [i]
    return rezult
}
```

Пример

Функция преобразования из десятеричной в двоичную форму. Функция to2() в этом примере принимает в качестве параметра десятеричное число и преобразует его в строку, содержащую это же число, но во двоичной форме. Эта строка состоит из нулей и единиц. Здесь используется рекурсивный вызов функции to2() самой себя. Обратите внимание, что использование ключевого слова var здесь принципиально важно из-за рекурсии, иначе функция будет работать неправильно.

```
function to2(n!0) {
    if (n!0 <2)
        return " " + n10              // чтобы результат был строковым
    var i=n!0%2
    var j=(nl0-i)/2
    return to2(])+i
}
```

Для преобразования строк, содержащих числа, в данные числового типа служат встроенные функции parselntQ и parseFloatQ соответственно для представления в целочисленном виде и в виде числа с плавающей точкой. Их мы уже рассматривали выше. Здесь следует отметить, что параметрами этих функций могут быть строки, содержащие числа не только в десятеричной, но и в шестнадцатеричнои, и в восьмеричной формах. Для указания основания системы счисления служит второй параметр этих функций.

Примеры

```
parseInt("ff",16)          // значение равно 255
parseInt("ff")             // значение равно NaN
parselnt("010")            // значение равно 8
parselnt (010", 8)         // значение равно 8
```

```
parseInt("010", 10)          // значение равно 10
parselnt("010", 2)           // значение равно 2
parselnt("010", 16)          // значение равно 17
```

Заметим, что функция parseFloatQ в IE6.0 работает неправильно, если указан второй параметр (основание системы счисления).

Для преобразования числа в строку, содержащую это число, достаточно использовать выражение сложения пустой строки с числом.

Примеры

```
"" + 25.78                  // значение равно "25.78"
"" + 2 . 5e3                // значение равно "2500"
```

Создание объекта Number

Числа можно создавать обычным образом с помощью переменных и оператора присвоения, не прибегая к объекту Number. Однако этот объект обладает некоторыми полезными свойствами и методами, которые иногда могут пригодиться.

Объект **Number** создается с помощью выражения вида:

переменная = new Number(число)

Доступ к свойствам и методам строкового объекта обеспечивается такими выражениями:

```
число.свойство
Number.свойство
число.методе[параметры])
Number,метод([параметры])
```

Свойства Number

- **MAX_VALUE** — константа, значение которой равно наибольшему допустимому в JavaScript значению числа (1.7976931348623157e+308).
- **MIN_VALUE** — константа, значение которой равно наименьшему допустимому в JavaScript значению числа (5e-324).
- NEGATIVEJNFINITY - число, меньшее, чем Number.MIN_VALUE.
- POSITIVEJNFINITY - число, большее, чем Number.MAX_VALUE.
- NaN — константа, имеющая значение NaN, посредством которой JavaScript сообщает, что данные (параметр, возвращаемое значение) не являются числами (Not a Number).
- prototype — свойство (прототип), играющее такую же роль, что и в случае объекта String (см. выше).

Методы Number

Объект Number имеет несколько методов, из которых мы рассмотрим только четыре, предназначенные для представления чисел в виде строки в том или ином формате.

1. toExponentiаl(KOflM4ecTBo) — представляет число в экспоненциальной форме.

   Синтаксис: **числоЛоЕхропепУаЦколичество)**

   Возвращает строку. Совместимость: IE5.5+, NN6+.

   Параметр представляет собой целое число, определяющее, сколько цифр после точки следует указывать.

Примеры

```
x=new Number(456)
x.toExponential(3)          //, 4.560e+2
x.toExponenttal(2)          // 4.56e+2
x.toExponential(1)          // 4.6e+2
x.toExponential(0)          // 5e+2
```

2. 1:oПxec1(количество) — представляет число в форме с фиксированным количеством цифр после точки.

   Синтаксис: числоЛоР1хес1(количество)

   Возвращает строку. Совместимость: IE5.5+, NN6+.

   Параметр представляет собой целое число, определяющее, сколько цифр после точки следует указывать.

   Примеры

```
x=new Number(25.65)
x.toFixed(3)          // 25.650
x.toFixed(2)          // 25.65
x.toFixed(1)          //25.7
x.toFixed(0)          // 25.7
```

3. **toPredsion(T04HocTb)** — представляет число с заданным общим количеством значащих цифр.

   Синтаксис: число.toPrecision(io4HocTb)

   Возвращает строку. Совместимость: IE5.5+, NN6+.

   Параметр представляет собой целое число, определяющее, сколько всего цифр, до и после точки, следует указывать.

   Примеры

```
x=new Number(135.45)
x.toPrecision(6)          // 135.450
x.toPrecision(5)          // 135.45
x.toPrecision(4)          // 135.5
x.toPrecision(3)          // 135
x.toPrecision(2)          // 1.4e2
x.toPrecision(l)          // Ie2
x.toPrecision(0)          // Сообщение об ошибке
```

4. t **1o51:ппд([основание])** — возвращает строковое представление числа в системе счисления с указанным основанием.

   Синтаксис: число.1:о5Мпд([основание])

   Возвращает строку. Если параметр не указан, имеется в виду десятеричная система счисления. Вы можете указать 2 для двоичной системы или 16 — для шестнадцатеричной. Заметим, что этот метод имеют все объекты.

   Примеры

```
x=new Number(127.18)
x.toStringO          // "127.18"
x.toString(10)          // "127.18"
x.toString(16)          // "7f.2el47ael47b"
x.toString(S)          //" 177.134121727024366"

x=new Number(5)
x.toString(2)          // "101"
```

1.7.4. Объект Math (Математика)

Объект Math предназначен для хранения некоторых математических констант (например, число я) и выполнения преобразований чисел с помощью типичных математических функций. Доступ к свойствам и методам объекта Math обеспечивается следующими выражениями:

```
Math.свойство
Math .метод(параметры)
```

Свойства Math

Свойства объекта Math имеют в качестве своих значений математические константы.

F	Постоянная Эйлера
LN10	Значение натурального логарифма числа 10
LN2	Значение натурального логарифма числа 2
LOG10E	Значение десятичного логарифма экспоненты (числа *в)*
LOG2E	Значение двоичного логарифма экспоненты
PI	Значение постоянной я
SQRT1_2	Значение квадратного корня ю-1/2
SORT	Значение квадратного корня из 2

Пример

Для вычисления длины окружности при известном радиусе требуется число п, которое можно взять как свойство объекта Math.

```
var R = 10                    // радиус окружности
circus = 2*R*Math. PI         // длина окружности
```

Методы Math

- aЬз(число) — возвращает модуль (абсолютное значение) числа;
- aco$(число) — возвращает арккосинус числа;
- a51п(число) — возвращает арксинус числа;
- atan(4HOio) — возвращает арктангенс числа;
- atan2(x, y) — возвращает угол в полярных координатах точки;
- cei(.(число) — округляет число вверх до ближайшего целого;
- co$(число) — возвращает косинус числа;
- exp(число) — возвращает число е в степени число;
- **поог(число)** — округляет число вниз до ближайшего целого;
- **1од(число)** — возвращает натуральный логарифм числа;
- **тах(число1,число2)** — возвращает большее из чиселчисло!, число2;
- **1тпп(число1,число2)** — возвращает меньшее из чиселчисло!, число2;
- **ро\л/(число!,число2)** — возвращаетч **исло!** в степени число2;
- **random()** — возвращает случайное число между 0 и 1;
- гоипс!(число) — округляет число до ближайшего целого;
- sin(число) — возвращает синус числа;
- sqrt(4Mcno) — возвращает квадратный корень из числа;
- 1ап(число) — возвращает тангенс числа.

Примеры

1. Метод randomQ возвращает случайное число, лежащее в интервале от 0 до 1. Чтобы получить случайное число в пределах от 0 до Nmax, следует написать следующее выражение:

   ```
   x = Nmax'Math . randomf)
   ```

   Если требуется получить случайное число в интервале от Nmin до Nmax, то из элементарного отношения пропорций получаем следующее выражение:

   ```
   x = Nmin + (Nmax - Nmi n) *Math. randomO
   ```

   Можно также создать функцию для вычисления случайного числа в заданном интервале:

   ```
   function rand(a, b) {
      return a+ (b-a) *Math . random()
   }
   ```

   Эта функция может потребоваться, например, для внесения некоторой непредсказуемости (нерегулярности) перемещения элементов на веб-странице, выбора цветов для мигающей надписи и т. п.

2. Для вычисления значения тригонометрической функции sin(x), у которой аргумент x выражен в градусах, следует применить следующее выражение:

   ```
   Math.sin(Math.PI*x/180)
   ```

Функции для решения некоторых математическихзадач

Язык JavaScript можно использовать не только для управления веб-страницами, но и для решения разного рода расчетных и математических задач. С другой стороны, составление или анализ соответствующих программ является хорошим упражнением в программировании. Поэтому предлагаю вам рассмотреть несколько несложных, но полезных программ расчетного характера.

Решение квадратного уравнения

С этой задачей мы знакомы еще со школы. Напомню, в чем она заключается. Требуется найти два числаxl и x_2 —.такие, чтобы при подстановке любого из них в выражение $ax^2 + bx + c$ результат его вычисления был бы равен 0. Такие x_1 и x_2 называются корнями уравнения $ax^2 + bx + c = 0$.

На первый взгляд может показаться, что для решения этой задачи следует просто вспомнить или найти в справочнике подходящую формулу. И действительно, это сделать нетрудно. Однако в программах для решения задач такого рода часто значительную долю кода занимают различные проверки и анализ исходных данных. Организация же вычислений по конечным формулам из справочника — не самое сложное. Так, в рассматриваемой задаче в зависимости от значений коэффициентов а, *А* и с возможны различные частные случаи. Корни могут быть комплексными и действительными. Допустим, нас интересуют только действительные корни. Тогда корней может и не быть совсем, или быть только один, или два, одинаковые или различные. Таким образом, прежде чем мы дойдем до применения формулы, придется проанализировать, с каким именно случаем имеем дело.

Очевидно, функция (назовем ее **beqQ)** для вычисления действительных корней квадратного уравнения должна принимать три параметра — коэффициенты *a, b* и с. Возвращаемым значением будет массив. Сделаем так, что если корней нет, то

этот массив будет пустым. В противном случае в нем будет два элемента. Если уравнение имеет два корня, то запишем их в качестве элементов возвращаемого массива. Если корень единственный, то он будет записан в первом элементе массива, а второй элемент будет пустым (null).

Ниже приводится один из возможных вариантов определения функции:

```
function beq(a, b, c){         // решение квадратного уравнения
var aret = newArrayO
var D = b*b - 4*a*c
if (a == 0) {
   if (!(b == 0)){
      aret[0] = -c/b
      aret[l] = null
   }
return aret                    // единственный корень или нет корней
}
   if (D == 0) {               // одинаковые корни
      aret[0]=-b/2/a
      aret[l]=aret[0]
   }
   if (D>0){                   // различные корни
      aret[0] = (-b - Math.sqrt(D))/2/a
      aret[l] = (-b + Math.sqrt(D))/2/a
   }
return aret
}
```

Выполним проверку работы функции на нескольких наборах исходных данных:

```
beq(0, 2, 6)            // массив (-3, null)
beq(l, -2, 1)           // массив (1, 1)
beq(3, 4, -2.5)         /* массив (-1.797054997187545,
                           0.4637216638542114) V
beq(2, 0. 5)            // пустой массив
```

Вычисление интеграла

Интеграл от некоторого выражения / *(x) с* одной переменной *x* на интервале ее значений от *a* до *b* можно понимать как площадь, ограниченную кривой *y =f(x),* осью абсцисс *(x)* и двумя прямыми, параллельными оси ординат *{y)* и проходящими через точки *a* и *b* на оси *(x).* Аналогия с площадью справедлива лишь отчасти. Если указать, что площадь кривой, проходящей ниже оси абсцисс, является отрицательной, тогда все в порядке.

Итак, для вычисления интеграла от выражения требуется вычислить площадь некоторой замысловатой фигуры. Алгоритм решения этой задачи основан на простой идее разбить сложную фигуру на множество простых, площадь которых легко вычислить, а затем взять сумму этих площадей. Здесь возникает вопрос о точности такого решения. Однако легко заметить, что чем меньше элементарные фигуры и чем больше их количество, тем ближе сумма их площадей к действительной площади исходной фигуры. Это так называемая задача приближения или аппроксимации.

В качестве элементарной фигуры можно взять прямоугольник или трапецию. Их площади легко вычисляются. Однако трапеция вписывается в кривую лучше, чем прямоугольник, поэтому выберем именно ее. Напомню: чтобы найти площадь трапеции, необходимо умножить полусумму ее параллельных сторон на высоту. Для

получения множества таких трапеций требуется разбить отрезок *aЪ* оси абсцисс на большое количество элементарных отрезков. В математике все идеально: отрезок *ab* разбивается на элементарные отрезки, длина которых стремится к нулю, то есть является бесконечно малой величиной. На практике мы имеем дело с конечными величинами. Более того, мы должны учесть, что с уменьшением длины элементарных отрезков и ростом их количества возрастает время вычислений. Таким образом, нам потребуется найти компромисс между точностью решения задачи и временными затратами.

Теперь займемся технической стороной вопроса. Что следует передавать нашей функции вычисления в качестве параметров? Я вспомнил о великолепной встроенной функции eval(), которая может вычислять выражения JavaScript, переданные ей в виде строки. Поэтому функция **integralQ** для вычисления интеграла будет принимать строку, содержащую выражение вида f(x), в котором переменная будет обозначена строчной латинской буквой x, например **"5*x*x** + 10". Два других параметра — числа а и Ь, соответствующие концам интервала интегрирования. Ниже приведено определение функции **integralQ:**

```
function integral(expression, a, b) {    // интеграл
var x, yl, y2 , n, length, dx, 5 = 0
length = Math.abs(b - a)
n = 100                          // количество элементарных отрезков
dx = length/n                    // длина элементарного отрезка
x = a                            /* начальное значение переменной x
                                    в выражении */
yl = eval(expression)
x = a + dx
y2 = eval(expression)
S = (yl + y2)'dx/2
for(i = 2; i< = n;l++){
   yl = У2
   x = x + dx
   y2 = eval(expression)
   S+= (yl + y2)*dx/2
}
return S
}
```

Проверим, как работает эта функция:

```
integral("x", 0, 1)         /* 0.5000000000000002
                               (точное значение = 0.5) */
integral("x*x" , 0, 1)      /* 0.33335000000000037
                               (точное значение = 0.3333...) */
integral("x*x" , 0, 10)     /* 333.34999999999917
                               (точное значение =333.3333333333...) */
```

Если точность вычислений нас не устраивает, то это обусловлено малым значением n количества отрезков разбиения интервала ab. При большой длине этого интервала n = 100 будет явно недостаточно. Конечно, можно передавать это значение функции в качестве четвертого параметра. Однако я поступил иначе: если длина интервала интегрирования больше 2, то n вычисляется по некоторой формуле, а иначе n равно 100. Кроме того, я добавил проверки крайних случаев, чтобы при них не возникало ошибок. В итоге получился следующий код:

```
function integral(expression, a, b){  // интеграл
if ((expression || !b&&!a)
```

```
    return 0
 if (a == b)
    return 0
var x, yl, y2, n, length, dx,S=0
length = Math.abs(b - a)
yl = Math.min(a.b)                  // если левый предел больше правого
b=Math.max(a,b)                      // если правый предел меньше левого
a = yl
п = 100
if (length>2) n=Math.round(100*Math.log(length+l))
dx = length/n                       // длина элементарного отрезка
x = a                               /* начальное значение переменной x
                                       в выражении */
yl = eval(expression)
x = a + dx
y2 = eval(expression)
S = (yl + y2)*dx/2
for(i=2; i<=n;i++){
   y l = y2
   x = x + dx
   y2 = eval(expression)
   S+= (yl + y2)*dx/2
}
return S
}
```

Выполним проверку:

```
integral("x*x", 0, 10)              /» 333.33749999999906
                                       (точное значение = 333.3333...) */
```

Заметьте, что в данном случае точность после наших коррекций увеличилась на порядок.

Вычисление производной

Производная от выражения (функции) / *(x) с* одной переменной *x* в некоторой точке *a* интерпретируется как скорость изменения значения этого выражения при значении *x,* равном *a.* Например, если выражение / *(x)* описывает зависимость пройденного пути от времени, то производная от/(x)в момент времени t равна скорости движения в этот момент.

Алгоритм решения этой задачи следующий: необходимо вычислить значения выражения в двух тестовых точках, расположенных рядом с заданной, а затем взять разность этих значений и разделить ее на расстояние между тестовыми точками. Чем ближе расположены тестовые точки к заданной, тем точнее получается результат. Мы выберем их так, чтобы заданная точка находилась посередине между тестовыми точками.

Как и вычисление интеграла, вычисление производной будет базироваться на встроенной функции evalQ, возвращающей значение выражения, переданное ей в виде строки как параметр. Функция DydxlQ для вычисления производной принимает строку, содержащую выражение, в котором переменная обозначена строчной латинской буквой x, например "25*x - 2". Второй параметр указывает точку, в которой следует вычислить значение производной. Вот вариант кода:

```
function Dydx(expression , a) {         // производная
if (( expression)                        // если нет выражения
```

```
  return 0
var x, y, dx
dx = 0.000025
x = a - dx                                    // 1-я тестовая точка
y = eval(expression)
x = a + dx                                    // 2-я тестовая точка
return (eval(expression) - y)/2/dx)
}
```

Проверочные примеры:

```
Dydx("x",1)            // 0.9999999999998899 (точное значение равно 1)
Dydx("x*x", 1)         // 1.9999999999997797 (точное значение равно 2)
```

Поиск экстремума

Экстремум выражения (функции)f(x) — это пара чисел $(x_0, г/_0)$, таких, что при подстановке^ в выражение/(л:) вместо переменнойxзначение этого выражения будет равно своему максимальному или минимальному значению уй. Вообще говоря, функция может иметь несколько экстремумов или не иметь их вообще. Мы напишем код функции **extremumQ,** которая вычисляет экстремум выражения, если он действительно у него есть (график этого выражения имеет «холм» или «впадину»).

Алгоритм решения задачи основан на том, что значение производной выражения в точке экстремума, или перегиба, равно 0.

Как и в случае вычисления интеграла и производной, при поиске экстремума будет использоваться встроенная функции eval(), возвращающая значение выражения, переданного ей в виде строки как параметр. Функция **extremumQ** для поиска экстемума принимает строку, содержащую выражение, в котором переменная обозначена строчной латинской буквой x, например "5*x*x - 2*x". Второй и третий параметры указывают границы интервала, в котором следует искать x_0. Последний, четвертый параметр задает точность поиска. Вот вариант кода:

```
function extremum(expression, a, b, d) {          // экстремум
if (!d) d = 0.001
var x, yl, y2 , i
x = a
yl = Dydx(expression, a)
i = 1
while ((x<= b)&&(i <= 1000)) {
y2 = Dydx(expression, x+d)
if ((yl<0)&&(y2>=0)||(y2<0)&&(yl>=0)) {
x=x+d*Math.abs(yl)/(Math.abs(yl)+Math.abs(y2))
return new Array(x, eval(expression))
)
x+= d
Y1 = Y2
i++
}
return new ArrayO
}
```

Проверочный пример:

```
extremum("x*x-l",-l, 1, 0.01)      /* (7.528699885739343e-16, -1)
                                      точное значение равно (0, -1) */
```

ВНИМАНИЕ

Если график выражения имеет несколько «холмов» или «впадин», то функция определит лишь один из них, ближайший к точке а.

1.7.5. Объект Date (Дата)

Во многих приложениях приходится отображать дату и время, подсчитывать количество дней, оставшихся до заданной даты, и т. п. Некоторые программы даже управляются посредством значений дат и времени. В основе всех операций, связанных с датами и временем, лежат текущие системные дата и время, установленные на вашем компьютере.

Со временем дела обстоят не так просто, как кажется на первый взгляд. Вспомните, что существуют временные зоны (часовые пояса), а также сезонные поправки времени. Так, например, текущее время в Санкт-Петербурге отличается от времени в Иркутске на 5 часов. Если в Иркутске уже полночь, то в Петербурге еще только 19 часов. Чтобы иметь возможность координировать деятельность во времени организаций и физических лиц в различных точках нашей планеты, была введена система отсчета времени. Она связана с меридианом, проходящим через астрономическую обсерваторию в городе Гринвич в Великобритании. Эту временную зону называют средним временем по Гринвичу (Greenwich Mean Time — GMT). Недавно кроме аббревиатуры GMT стали использовать еще одну — UTC (Universal Time Coordinated — Всеобщее Скоординированное Время).

Если системные часы вашего компьютера установлены правильно, то отсчет времени производится в системе GMT. Однако на Панели управления обычно устанавливается локальное время, соответствующее вашему часовому поясу. При создании и изменении файлов на вашем компьютере фиксируется именно локальное время. Вместе с тем операционная система знает разницу между локальным временем и GMT. При перемещении компьютера из одного часового пояса в другой необходимо изменить установки именно часового пояса, а не текущего системного времени (показания системных часов). Даты и время, генерируемые в сценариях, сохраняются в памяти в системе GMT, но пользователю выводятся, как правило, в локальном виде.

В программе на JavaScript нельзя просто написать 30.10.2002, чтобы получить значение даты, с которым в дальнейшем можно производить некие операции. Значения даты и времени создаются как экземпляры специального объекта Date. При этом объект сам будет «знать», что не бывает 31 июня и 30 февраля, а в високосных годах 366 дней.

Создание объекта даты

Объект даты создается с помощью выражения вида:

```
имяОбъектаДаты = new Date([параметры])
```

Параметры не обязательны, на что указывают квадратные скобки. Обратите внимание, что имяОбъектаДаты является объектом даты, а не значением какого-нибудь другого типа (например, строкой или числом).

Для манипуляций с объектом даты применяется множество методов объекта Date. При этом используется такой синтаксис:

```
переменная = имяОбъектаДаты.методО
```

Если, например, объекту даты требуется присвоить новое значение, то для этого используется соответствующий метод:

```
переменная = имяОбъектаДаты.метод(новое_значение)
```

Рассмотрим в качестве примера изменение значения года текущей системной даты:

```
xdate = new DateQ()          /* создание объекта, содержащего
                                текущую дату и время */
Year = xdate.getYear()()     /* в переменной Year содержится
                                значение текущего года */
   Year = Year +33           /* в переменной Year содержится
                                значение,большее, чем текущий год, на 3 */
   xdate.setYear(Year)        /* в объекте устанавливается
                                новое значение года */
```

При создании объекта даты с помощью выражения new DateQ, можно указать в качестве параметров, какие дату и время следует установить в этом объекте. Это можно сделать пятью способами:

```
new Datef Месяц дд, гггг чч:мм:сс")
new Оаг.е("Месяц дд, гггг")
new Date(rr, мм, дд, чч, мм, ее)
newDate(rr, мм,дд)
new Оат.е(миллисекунды)
```

В первых двух способах параметры задаются в виде строки, в которой указаны компоненты даты и времени. Буквенные обозначения определяют шаблон параметров. Обратите внимание на разделители — запятые и двоеточия. Время указывать не обязательно. Если компоненты времени опущены, то устанавливается значение 0 (полночь). Компоненты даты обязательно должны быть указаны. Месяц указывается в виде полного его английского названия (аббревиатуры не допускаются). Остальные компоненты указываются в виде чисел. Если число меньше 10, то можно писать одну цифру, не записывая ведущий 0 (например. 3:05:32).

В третьем и четвертом способах компоненты даты и времени представляются целыми числами, разделенными запятыми.

В последнем способе дата и время задаются целым числом, которое представляет количество миллисекунд, прошедших с начала 1 января 1970 года (то есть с момента 00:00:00). Количество миллисекунд, отсчитанное от указанной стартовой даты, позволяет вычислить все компоненты и даты, и времени.

Методы объекта Date

Для чтения и изменения информации о дате и времени, хранящейся в объекте даты, служат методы объекта Date (табл. 1.3). Напомним, что объект даты создается с помощью выражения:

```
имяОбъектаДаты = new Date([параметры])
```

Затем, чтобы применить метод метод() к объекту даты имяОбъектаДаты, следует написать: имяОбъектаДаты.метод([параметры]).

Довольно большое множество всех методов можно разделить на две категории: методы получения значений (их названия имеют префикс get) и методы установки новых значений (их названия имеют префикс set). В каждой категории выделяются две группы методов — для локального формата и формата UTC. Методы позволяют работать с отдельными компонентами даты и времени (годом, месяцем, числом, днем недели, часами, минутами, секундами и миллисекундами).

Таблица 1.3. Методы объекта Date

Метод	Диапазон значений	Описание
getFuUYear()	1970-...	Год
getYear()	70-...	Год
getMonth()	0-11	Месяц (январь = 0)
getDate()	1-31	Число
getDay()	0-6	День недели (воскресенье = 0)
getHours(	0-23	Часы в 24-часовом формате
getMinutes()	0-59	Минуты
getSeconds()	0-59	Секунды
getTime()	0-...	Миллисекунды с 1.1.70 00:00:00 GMT
getMHliseconds()	0-...	Миллисекунды с 1.1.70 00:00:00 GMT
getUTCFuLYear()	1970-...	Год UTC
getUTCMonth()	0-11	Месяц UTC (январь = 0)
getUTCOate()	1-31	Число **UTC**
getUTCDay()	0-6	День недели UTC (воскресенье = 0)
getUTCHours()	0-23	Часы UTC в 24-часовом формате
getUTCMinutes()	0-59	Минуты UTC
getUTCSecondsQ	0-59	Секунды UTC
getUTCMilliseconds()	0-....	Миллисекунды UTC с 1.1.70 00:00:00 GMT
setYear(3H34)	1970-...	Установка года (четырехзначного)
SetFuUYear(3H34)	1970-...	Установка года
setMonth(3Ha4)	0-11	Установка месяца (январь = 0)
setDate(3H34)	1-31	Установка **числа**
setDay(3H34)	0-6	Установка дня недели (воскресенье = 0)
setHours(3H34)	0-23	Установка часов в 24-часовом формате
setMinutes(3H34)	0-59	Установка минут
setSeconds(3H34)	0-59	Установка секунд
setMiLliseconds(3Ha4)	0-...	Установка миллисекунд с 1.1.70 00:00:00 GMT
setTime(3H34)	0-...	Установка миллисекунд с 1.1.70 00:00:00 GMT
setUTCFuLLYear(3H34)	**1970-...**	Установка года UTC л

— — — *проиолжепие ту*

Таблица 1.3 *(продолжение)*

Метод	Диапазон значений	Описание
setUTCMonth(3H34)	0-11	Установка месяца UTC (январь = 0)
setUTCD3te(3H34)	1-31	Установка числа UTC
setUTCDay(3H34)	**0-6**	Установка дня недели UTC (воскресенье = 0)
setUTCHours(3Ha4)	**0-23**	Установка часов UTC в 24-часовом формате
setUTCMinutes(3H34)	0-59	установка минут UTC
setUTCSeconds(3Ha4)	0-59	Установка секунд UTC
setUTCMilliseconds(3H34)	0-...	Установка миллисекунд UTC с 1.1.70 00:00:00 **GMT**
getTimezoneOffset ()	**0-...**	Разница в минутах по отношению к **GMT/UTC**
toDateString()		Строка с датой (без времени) в формате браузера (IE5.5)
toGMTString()		Строка с датой и временем в глобальном формате
toLoc3leDateString()		Строка с датой без времени в локализованном формате системы (NN6, **IE5.5)**
toLocaleString()		Строка с датой и временем в локализованном формате системы
toLocaleTimeString()		Строка с временем без даты и в локализованном формате системы **(NN6,** IE5.5)
toString()		Строка с датой и временем в формате браузера
toTimeString()		Строка с временем без даты в формате браузера (IE5.5)
toUTCString()		Строка с датой и временем в глобальном формате
D3te.parse("dateString")		Преобразование строки с датой в число миллисекунд
Date.UTC(3H34)		Преобразование строки с датой GMT в число

Вычислять разность двух дат или создавать счетчик времени, оставшегося до некоторого заданного срока, можно с помощью методов как для локального формата, так и для UTC. Не следует применять выражения, использующие различные форматы времени, поскольку результаты могут оказаться неправильными. Формат UTC обычно применяется в расчетах, учитывающих часовой пояс.

Следует также учитывать, что нумерация месяцев, дней недели, часов, минут и секунд начинается с 0. Для компонентов времени это естественно. Однако при такой нумерации декабрь оказывается 11-м месяцем в году, а не 12-м, как это принято повсеместно. Воскресенье (Sunday) является 0-м днем недели, а не 7-м.

Значение года XX века представляется в двузначном формате как разность между этим годом и 1900. Например, 1998 год представляется как 98. Годы до 1900 и после 1999 обозначаются в четырехзначном формате. Например, 2002 год нуж-

нотаки писать —2002, поскольку02 — это 1902 год. МетодgetFullYear() возвращает четырехзначное значение года.

Примеры

Допустим, что сейчас 2003 год.

```
today = new Date()             // текущие дата и время
Year = today.getYearO          // 2003
today = new Date(98,11,6)      /* объект даты today содержит информацию
                                  о дате 6 ноября 1998 года, 00:00:00 */
Year = today.getYearO          // 98
today.getFullYearO             // 1998
```

Изменение любого компонентадаты производится с помощью 'соответствующего метода, название которого начинается с приставки set. При этом значения других компонентов пересчитываются автоматически.

В следующем примере мы создаем объект даты, содержащий некоторую конкретную дату. Затем мы устанавливаем новое значение года. При этом в объекте даты изменяется и число.

```
mydate = new Date(1952, 10,6)   // "Thu Nov 6 00:00:00 UTC+0300 1952"
myday = mydate. getDayO         // 6
mydate.setYear(2003)            // установка 2003 года
myday = mydate .getDayO         // 4
```

При попытке отобразить значение объекта даты оно автоматически преобразуется в строку методом **toString().** Формат этой строки зависит от операционной системы и браузера. Например, для Windows 98 и Internet Explorer 5.5 строка, содержащая информацию объекта даты, имеет следующий вид:

```
Thu Oct 31 13:16:23 UTC+0300 2002,
```

то есть четверг, октябрь, 31, 13 часов 16 минут 23 секунды всеобщего времени, скорректированного на 3 часа, 2002 год.

Если коррекцию времени с учетом часового пояса производить не требуется, то для строкового представления даты и времени можно воспользоваться методом **toLocaleString():**

```
mydate = new DateO
mydate.toLocaleStringO          // "31 октября 2002 г. 14:15:30"
```

В браузерах **1E5.5+ И N N 6 +** работают еще два метода представления отдельно даты и времени в виде строки:

```
mydate = new DateO
mydate. toLocaleDateStringO     // "31 октября 2002 г."
mydate.toLocaleTimeString()     // "14:15:30"
```

Заметим, что формат представления даты и времени, обеспечиваемый методами **toLocalDateStringO** и **toLocalTimeStringQ,** зависит от настроек операционной системы и браузера. Если вы будете использовать эти методы для вывода даты и времени на веб-страницу, то они будут выглядеть так, как пользователь привык их видеть на своем компьютере.

С датой иногда приходится выполнять различные вычисления, такие как определение даты через заданное количество дней от текущей или количество дней между двумя датами. Вот здесь как раз и требуется предварительный подсчет миллисекунд, содержащихся в минуте, часе, сутках и т. д.

Примеры

1. Определим дату, которая наступит через неделю относительно текущей:

```
week = 1000*60*60*24*7              /* количество миллисекунд
                                       в неделе - 604800000 */
mydate = new Date()                 // объект даты с текущей датой
mydate_ms = mydate.getTimeQ         /* текущая дата, представленная
                                       количеством миллисекунд
                                       от 1.01.1970 00:00:00 */
mydate_ms += week                   // maydate_ms = mydate_ms + week
mydate.setTime(mydate_ms)           /* установка новой даты
                                       в объекте mydate */
newdate = mydate.toLocaleStringO    // новая дата в виде строки
```

2. Определим количество дней между двумя датами, например 10 февраля 2003 и 5 марта 2003. Для этого создадим сначала два соответствующих объекта даты:

```
datel = new, Date(2003 ,01,10)
date2 = new Date(2003,02,5)
```

Переменные datel и dateZ содержат длинные строки, содержащие даты (например, Mон Feb 10 00:00:00 UTC+0300 2003). Чтобы перевести их в количество миллисекунд, воспользуемся методом parse() объекта Date. Затем вычислим разность **Date.parse(date2)** и **Date.parse(datel)** и разделим ее на количество миллисекунд в одних сутках:

```
days = (Date.parse(date2) - Date.parse(datel))/1000/60/60/24
                                    // результат: 23
```

3. Часто приходится иметь дело с датами, представленными в виде строк в формате **"дд.мм.ггг"**, причем месяцы нумеруются с 1 до 12, а не с 0 до 11. В этом случае, чтобы воспользоваться методами объекта Date, необходимо предварительно выполнить соответствующие преобразования.

Допустим, исходная дата strdate представляется в виде строки в формате "дд.мм.гггг", а для создания объекта даты используется формат параметров "г, мм, дд". Тогда необходимые преобразования исходных данных выглядят следующим образом:

```
astrdate = strdate.split(".")   /* массив подстрок, полученных
                                   из строки strdate с использованием
                                   точки как разделителя */
                                /* создаем объект даты */
mydate = new Date(astrdate[2] , astrdate[1]-1, astrdate[0])
```

4. Развивая идею, рассмотренную в предыдущем примере, мы могли бы создать функцию, принимающую в качестве параметра строку, содержащую дату и время в привычном для нас формате **"дд.мм.гг чч:мм:сс"** и возвращающую объект даты. Будем считать, что в строке параметра нашей функции дата отделена от времени одним пробелом, однако допускается, что информации о времени вообще может не быть. Далее, если время присутствует, секунды не обязательно должны указываться. Если параметр является пустой строкой или вообще отсутствует, то функция возвращает объект с текущей датой.

```
function mydate(strdate){
if (strdate == ""|| strdate == null) {
   return new Date()
}
var astrdate = strdate.split(" ")
```

```
if (astrdate. length == 1)
   astrdate[l] = "00:00:00"
astrdate[0] = astrdate[0].split(".")
astrdatefl] = astrdate[l].split(":")
if (astrdate[l] .length == 1)
   astrdate[l] = new Array (astrdate [1] [0] , "00", "00")
else {
   if (astrdate[l] .length == 2)
      astrdatefl] = new Array (astrdate [1][0] , astrdate [1][1] , "00")
}
return new Date(astrdate[0][2], astrdate[0] [1]-1, astrdate[0] [2] ,
astrdate[l][0]. astrdate [1] [1] , astrdate[1] [2])
}
```

Заметим, что в действующем коде выражение с последним оператором return необходимо написать в одну строку.

С использованием функции **mydateQ** количество дней между двумя заданными датами можно вычислить следующим образом:

```
datel = mydate("10.2.2003")
date2 = mydate("5 . 3 . 2003")
days = (Date.parse(date2) - Date.parse(datel))/1000/60/60/24
```

Иногда в приложениях, в том числе и на веб-страницах, устанавливают часы (таймер), показания которых постоянно изменяются в соответствии с ходом системных часов. Чтобы сделать это, требуется в цикле создавать объект даты и формировать строку из компонент времени, используя подходящие для этого методы. Однако при этом программа, в которой имеется код, выполняющий роль часов, будет заниматься только этими часами (зациклится). Чтобы этого не произошло, используют метод **setlntervalQ** из объектной модели браузера. Позднее мы рассмотрим задачу создания часов подробно.

1.7.6. Объект Boolean (Логический)

Объект Boolean создается с помощью выражения вида:

```
переменная = new Boo1eaп(логическое_значение)
```

Он имеет свойство prototype, методы **toStri**ngQ и значение Of (), которые имеют также объекты String и Number.

Смысл свойства prototype мы уже рассматривали применительно к объектам String и Array. Объект Boolean может понадобиться в том случае, когда всем логическим объектам, создаваемым с помощью выражения с ключевыми словами new Boolean, нужно добавить новые свойства или методы с помощью прототипа (свойства prototype).

1.7.7. Объект Function (Функция)

Создание объекта Function

Выше мы уже рассматривали стандартный способ определения функции:

```
function имя_функции(параметры) {
      код
}
```

Существует и другой способ, основанный на выражении с ключевыми словами new Function. Согласно этому способу функция создается как экземпляр объекта Function:

```
имя_функции = new Function(["napl", [."napN"]. "оператор!; [;
onepaTopN]")
```

Названия всех параметров являются строковыми значениями. Они разделяются запятыми. Заключительная строка содержит операторы кода тела функции, разделенные точкой с запятой.

Вызов функции, определенной как экземпляр объекта Function, можно выполнить обычным способом: имя_функции(параметры).

Примеры

```
Srectangle = new Function("width", "height", "var s = width'height;
return s")
Srectangle(2, 3)                                    // возвращает 6

var expr = "var s = width*height; return s"
Srectangle = new Function("width", "height", expr)
Srectangle (2 , 3)                                  // возвращает 6

a = "width"
b = "height"
expr = "var s = width*height; return s"
Srectangle = new Function(a, b, expr)
Srectangle(2, 3)                                    // возвращает б
```

При любом задании функции, стандартном или с помощью ключевого слова new, автоматически создается экземпляр объекта Function, который обладает своими свойствами и методами.

Свойства Function

1. arguments — массив значений параметров, переданных функции.

   Индексация элементов массива производится с 0. Поскольку это массив, он имеет свойства и методы объекта Array (в частности, свойство length — длина массива).

   Свойство arguments применяется в теле определения функции, когда требуется проанализировать параметры, переданные ей при вызове. Например, можно узнать, сколько в действительности было передано параметров, не являются ли их значения пустыми ("", 0, null) и т. п. Это свойство особенно полезно при разработке универсальных библиотечных функций.

   Синтаксис выражения следующий: имя_функции.arguments.

   Пример

   Функция, приводимая в этом примере, возвращает строку, содержащую значения параметров и их общее количество, которые были указаны в вызове функции (а не в ее определении!).

```
function myfunc(a, b,c){
var arglenth=myfunc.arguments.length      /* количество переданных
                                             параметров */
var x=""
for (i=0; i< myfunc.arguments.length;i++) {
```

```
x += myfunc.arguments[i] + ","
}
return x+"Всего: "+ myfunc.arguments.length
)
myfunc(5, "Вася")                     // "5, Вася, Всего: 2"
myfuncO                                // "Всего: 0"
myfunc(null,"" ,0,25)                 // "null , ,0,25 , всего: 4"
```

2. Length — количество параметров, указанных в определении функции.

   Синтаксис: имя_функции. length

   В отличие от свойства arguments, количество параметров функции можно определить в программе за пределами тела этой функции.

   Пример

```
function myfunc(a, b, c, d){
return myfunc.arguments.length
.)
myfunc(a.b)                           // 2
myfunc. length                        // 4
```

3. caller — содержит ссылку на функцию, из которой была вызвана данная функция; если функция не вызывалась из другой функции, то значение этого свойства равно null.

   Совместимость: IE4+, NN4.

   Синтаксис: имя_функции. caller

   В свойстве имя_функции.caller содержится все определение функции, из которой была вызвана функция имя_функции.

   Пример

```
function fl() {
f2()
}
function f2()
alert(f2 . caller)   // диалоговое окно со значением f2.caller
}
f K)
```

   Вызов функции fl() приведет к вызову функции f2(), которая выведет на экран окно с сообщением, содержащим код определения функции fl() (рис. 1.12).

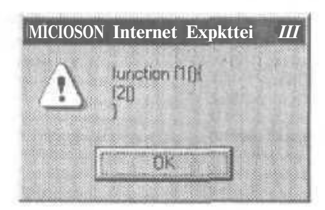


Рис. 1.12. Результат выполнения функции fl() — окно со значением f2.caller

Методы Function

toString() — возвращает определение функции в виде строки.

Синтаксис: имя_функции.1о5Ылд()

Иногда этот метод используют в процессе отладки программ с помощью диалоговых окон.

Пример

```
function myfunc(a, b) {
return a*b/2
I
alert (myfunc. toStringO)
```

Результат выполнения последнего выражения показан на рис. 1.13.

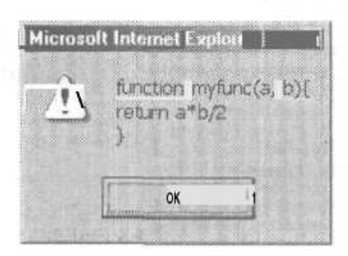


Рис. 1.13. Результат выполнения alert(myfunc.toString())

```
apply( [текущий_объект [, массив_параметров]])
call( [текущий_объект [ ,napl [, пар2 [ . . . napN] ] ] ])
```

Оба метода используются для вызова функции и дают одинаковые результаты. Отличаются они лишь формой представления параметров. Как известно, функцию можно вызвать просто по имени, за которым следует список параметров в круглых скобках. Особенностью этих методов является то, что с их помощью можно вызвать функцию по ссылке на нее. При первом чтении мы рекомендуем сначала ознакомиться с разделом 1.8, а. затем вернуться к данной теме, если в этом возникнет необходимость.

Первый параметр обоих методов — ссылка на объект, являющийся текущим для данной функции. Ссылка на объект используется, когда вызываемая функция определена как метод пользовательского объекта. Более подробно эта тема обсуждается ниже, в разделе 1.8.

Пример

```
/* Функция-конструктор объекта car (автомобиль)*/
function car(name, model, color) {
this.name = name                    // название
this.model = model                  // модель
his.color = color                   // цвет
this.show = showcar                 // метод
}
/* Функция, вызываемая как метод объекта car */
function showcar(){
alert(this.name + " - " + this.model)     // окно с сообщением
:i
/* Создание экземпляра объекта car */
• mycar = new car("Жигули", "BA32105","white")
```

Обычный способ применения метода show к объекту car (вывод окна с сообщением) выглядит так:

mycar.show()

При этом в действительности вызывается функция **showcarQ,** заданная для этого метода. Текущий объект car рассматривается как контекст для ссылок **this,** указанных в теле функции.

Однако методы apply() и callQ позволяют не связывать функцию **showcarQ** с объектом mycar. Более того, в конструкторе объекта сагможно не указывать выражение, определяющее функцию **showcar** в качестве метода объекта. Вместо этого можно вызвать метод showcar как объект и применить к нему метод **callQ,** указав в качестве параметра mycar как текущий объект:

```
showcar.call(mycar)
```

С помощью метода **catlQ** можно передавать параметры, отделенные друг от друга запятыми. Если параметры определены как элементы массива, то вместо саЩ) используется метод **applyQ.**

1.7.8. Объект Object

Object является корневым объектом, на котором базируются все остальные объекты **JavaScript,** такие как String, Array, Date и т. д. В программах можно создавать свои собственные объекты. Это можно сделать различными способами.

Способ 1

```
function имя_конструктора ([napl,...[, napN]){
код
>
имяОбъекта = new имя_конструктора(["napl",...[, "napM"])
```

Способ 2

```
имяОбъекта = new Object()
имяОбъекта.свойство = значение
```

Способ 3

```
имяОбъекта = {свойство!: значение! [, свойство2: значение2 [ N ] }
```

Для обращения к свойствам и методам объекта используется следующий синтаксис:

```
ссылка_на_объект.свойство
ссылка_на_объект.метод([параметры])
```

Допустим, например, что нам требуется создать объект Сотрудник, который содержал бы сведения о сотрудниках некоторой фирмы, такие как Имя, Отдел, Телефон, Зарплата и т. п. В фирме может быть много сотрудников, но сведения о них представляются в некоторой единой структуре. Эту структуру можно создать с помощью конструктора объекта:

```
function Сотрудник(Имя, Отдел, Телефон, Зарплата) {
this.Имя = Имя
this.Отдел = Отдел
this.Телефон = Телефон
this.Зарплата = Зарплата
}
```

Как видите, конструктор объекта может быть представлен в виде определения функции. Ключевое свойство this представляет ссылку на текущий, то есть определяемый конструктором объект. Все операторы присвоения с this, расположенные в теле функции-конструктора, определяют свойства объекта. В круглых скобках у имени объекта могут перечисляться параметры, чтобы иметь возможность

создать конкретный объект, являющийся экземпляром обезличенного объекта Сотрудник, и присвоить его свойствам конкретные значения. Например, создадим конкретного сотрудника — agentOO?:

```
agentOO? = new Сотрудник("Джеймс Бонд", 5, "223-332", 3600.50)
```

Доступ к свойствам этого объекта производится обычным способом:

```
a§ет007.Имя                    // "Джеймс Бонд"
agent007.Зарплата              // 3600.5
```

Информация о новом сотруднике добавляется аналогичным образом:

```
Shtirlitz = new Сотрудник("Максимов", 4, "123-4567", 4500.50)
```

К свойствам и методам этого объекта обращаются довольно редко, и здесь мы не будем их подробно рассматривать. Отметим свойство prototype, назначение которого мы уже описывали применительно к объектам String и Array. Метод toStringQ обычно используется при отладке программ с помощью диалоговых окон. Свойство hasOwnPropery("cBoiicTBo") используется, чтобы определить, имеется ли у экземпляра объекта указанное свойство, определенное в его конструкторе (но не с помощью prototype). Если да, то возвращается true, в противном случае — false.

Более подробные сведения о создании объектов вы найдете в следующем разделе. Там приведено много примеров, и вы легко убедитесь, что понятие объекта не сложнее, чем понятие функции.

1.8. Пользовательские объекты

В предыдущем разделе мы рассмотрели встроенные объекты, то есть заранее предопределенные в JavaScript и часто используемые в программах. С помощью выражений с ключевым словом new вы можете создавать экземпляры этих объектов, то есть их конкретные воплощения. Более того, благодаря свойству prototype имеется возможность добавлять к объектам новые свойства и методы, придуманные пользователем и отсутствовавшие в исходных встроенных объектах. В большинстве случаев, в частности при создании сценариев для веб-страниц, всего этого более чем достаточно. Однако нельзя обойти вниманием возможность создания собственных объектов. Зачем нужны собственные объекты?

Они не являются необходимыми для решения практических задач. С точки зрения программиста, объект представляет собой просто удобное средство организации данных и функций их обработки. Чтобы как-то организовать данные и функции в программе, далеко не всегда необходимо прибегать к такой конструкции, как объект. Ведь даже при внушительном количестве данных программист не всегда организует их в виде массива (объекта Array): бывает достаточным ограничиться простыми переменными.

В этом разделе мы сосредоточимся на том, как создавать объекты, если возникла такая необходимость. Возможно, это вам понадобится. Кроме того, данный материал поможет лучше понять природу встроенных объектов, а также общую философию объектно-ориентированного программирования. Сейчас концепция объектно-ориентированного программирования является ведущей в технологиях создания крупных приложений, поэтому будет полезно познакомиться с ней поближе.

1.8.1. Создание объекта

Объекты в JavaScript можно создать несколькими способами. Мы рассмотрим три из них.

Первый способ основан на функции, в теле которой описываются все свойства и методы создаваемого объекта. Поскольку эта функция играет определяющую роль в создании объекта, ее называют функцией-конструктором или просто конструктором объекта. Что, собственно, должен делать конструктор? Очевидно, он должен ввести имя создаваемого объекта, а также его свойства и методы. Кроме того, он должен допускать возможность присвоения начальных значений свойствам.

Имя функции-конструктора объекта является одновременно и именем создаваемого объекта. Свойства и методы создаваемого объекта задаются в теле функции-конструктора с помощью операторов присвоения. При этом имена переменных-свойств записываются с ключевым словом this (этот): this.переменная.

Рассмотрим в качестве примера функцию-конструктор, определяющую объект car (автомобиль) со свойствами name (название), model (модель) и color (цвет):

```
function car(name, model, color) {
   this.name = name
   this.model = model
   this.color = color
}
```

Эта функция определяет объект car. Чтобы создать конкретный экземпляр объекта car, следует выполнить выражение с вызовом этой функции, которой можно передать значения параметров. Мы уже знаем, что экземпляр объекта создается с помощью оператора присвоения с ключевым словом new:

```
mycar = new car("Ока", "ВА31111", "white")
```

Итак, мы создали объект mycar, являющийся экземпляром объекта car. Таких экземпляров с различными именами можно создать несколько. Значения свойств объекта mycar можно изменять в программе:

```
mycar.model                  // значение равно "ВА31111"
mycar.name                   // значение равно "Ока"
mycar.model = "ВА311113"     // значение равно "ВА311113"
```

Как видите, объект — это просто особый способ группировки данных и их использования (составное имя переменной: объект.свойство).

Объекты можно создавать и с помощью конструктора new ObjectQ:

```
mycar = new ObjectO
mycar.name = "Ока"
mycar.model = "ВА31111"
mycar.color = "white"
```

В этом способе отчетливо видно: создаваемый объект является экземпляром объекта Object, подобно тому как, например, создаваемый массив является экземпляром объекта Array.

Допускается также и следующая компактная запись определения объекта:

```
mycar = {пате:"Ока", model:"ВА3111", color:"white"}
```

Здесь все определение свойств заключается в фигурные скобки. Пары свойство= =значение разделяются запятыми, а имя свойства отделяется от значения двоеточием.

При создании объекта мы можем задать значения свойств по умолчанию, то есть значения, которые будут иметь свойства, если при создании экземпляра этого объекта значения его свойств не указаны явным образом (то есть имеют значения null, 0 или ""). Это делается с помощью логического оператора ИЛИ (обозначаемого ||) в конструкторе объекта в виде функции:

```
function car(name, model, color) {          // конструктор объекта car
   this.name = name || "неизвестно"
   this.model = model || "неизвестно"
   this.color = color || "black"
}

mycar = new car("Жигули","","")  // создание экземпляра mycar объекта car
   mycar.name                    // "Жигули"
   mycar.model                   // "неизвестно"
   mycar.color                   // "black"
```

1.8.2. Добавление свойств

Если возникает необходимость добавить новое свойство к существующему объекту, а также ко всем его экземплярам, то это можно сделать с помощью свойства prototype. В приведенном ниже примере мы создаем объект car (автомобиль), затем его экземпляр mycar, а затем добавляем к свойству prototype свойство owner (владелец) с конкретным значением:

```
function car(name, model, color) {    // конструктор объекта car
   this.name = name || "неизвестно"
   this.model = model || "неизвестно"
   this.color = color || "black"
}

mycar = new car("Жигули","","")          /* создание экземпляра mycar
                                            объекта car */
mycar.name                               // "Жигули"
mycar.model                              // "неизвестно"
mycar.color                              // "black"
car.prototype.owner = "Иванов"           // добавляем новое свойство
mycar.owner                              // "Иванов"
```

Если нужно добавить новое свойство только к конкретному объекту (к данному экземпляру объекта), то это можно сделать просто с помощью оператора присвоения:

```
имяОбъекта.новое_свойство = значение
```

В следующем примере мы создаем объект car вообще без свойств, а затем добавляем его экземпляры и к ним — различные свойства.

```
function car(){}
   mycar1 = new car()
   mycar2 = new car()
   mycar1.name = "Жигули"
   mycar2.model = "ВАЗ2106"

   mycar1.name                // "Жигули"
   mycar1.model               // undefined (не определено)
   mycar2.name                // undefined (не определено)
   mycar2.model               // "ВАЗ2106"
```

1.8.3. Связанные объекты

В объекте в виде свойства может содержаться ссылка на другой объект. В этом случае оба объекта оказываются связанными: один из них оказывается подобъектом или, другими словами, свойством другого. Например, мы можем создать объект photo, содержащий в качестве своих свойств название автомобиля и URL-адрес файла с его изображением. Такой объект можно связать с объектом car, содержащим основную информацию об автомобилях, добавив к нему свойство, значением которого является ссылка на объект photo.

Ниже приведены конструкторы объектов car и photo. Чтобы добавить объект photo в объект **car,** мы добавили в объект **car** свойство, значением которого является ссылка на объект photo.

```
function car(name, model, color, photo) {    // конструктор объекта car
   this.name = name
   this.model = model
   this.color = color
   this.photo = photo                           // ссылка на объект photo
}

function photo (name, url) {                    // конструктор объекта photo
   this.name = name
   this.url = url
}
```

Создадим пару конкретных объектов, являющихся экземплярами объекта photo:

```
photo1 = new photo("Ока", "/images/oka1.jpg")
photo2 = new photo("Ока", "/images/oka2.gif")
```

Теперь создадим экземпляры объекта car:

```
mycar = new car("Ока", "ВА31111", "white", photo1)
mycar.photo.url                                // "/images/oka1.jpg"
mycar.model                                    // "ВА31111"

mycar = new car("Ока", "ВА31111", "white", photo2)
mycar.photo.url                                // "/images/oka2.gif"
mycar.model                                    // "ВА31111"
```

Обратите внимание: чтобы узнать значение URL-адреса расположения файла с картинкой, относящейся к объекту mycar, мы обращаемся к свойству url объекта photo, который сам является свойством объекта mycar или, другими словами, подобъектом объекта mycar.

1.8.4. Пример создания базы данных с помощью объектов

Теперь рассмотрим пример создания базы данных автомобилей, находящихся на одной стоянке. Список всех автомобилей можно представить в виде таблицы со столбцами: name (название), model (модель), **regnum** (номер), owner (владелец) и photo (фотография). Будем считать, что на фотографии номер автомобиля не показан, так что она отображает не конкретный автомобиль, а его тип (название, модель и цвет). Конечно, можно создать базу данных в виде одной таблицы, однако мы поступим иначе и создадим две таблицы. Одна из них будет вспомогатель-

ной и выполнять роль справочника типов автомобилей, а другая таблица будет содержать записи о конкретных автомобилях, находящихся на стоянке или, точнее, приписанных к ней.

Справочник типов автомобилей будет содержать наиболее общие сведения об автомобилях: название, модель и фотографию (точнее, URL-адрес ее файла). На стоянке может находиться несколько автомобилей одного типа (например, «Жигули», ВА32101 белого цвета), но в справочнике этот тип будет упомянут лишь один раз. В нашей базе данных справочник типов автомобилей будет представлять объект **ref**:

```
function ref(name, model, url) {        /* конструктор справочника
                                           типов автомобилей */
   this.name = name                     // название марки автомобиля
   this.model = model                   // модель
   this.url = url                       // URL-адрес файла с фотографией
}
```

Заполним справочник конкретными записями. Множество всех таких записей удобно определить как массив (назовем его aref). Тогда элементы этого массива определим как экземпляры объекта ref:

```
aref = new ArrayO        // массив записей справочника типов автомобилей
aref[0] = new ref("Жигули",, "Ваз2101", "pict0.gif")
aref[l] = new ref("Жигули", "Ваз2106", "pictl.gif")
aref[2] = new ref("Bcwira", "ГА324", "pict2.gif")
aref[3] = new ref("OKa", "Baslll l", "pict3.gif")
```

В нашем справочнике описаны только четыре типа автомобилей.

Теперь создадим список всех автомобилей на стоянке. Этот список удобно оформить как массив, каждый элемент которого представляет запись о конкретном автомобиле. Формально элементы этого массива мы определим как экземпляры объекта car, который будет содержать ссылку на справочник — объект ref.

Итак, сначала напишем код функции-конструктора объекта **car**:

```
function car(regnum, owner, ref) {
   this.regnum = regnum             // номер автомобиля
   this.owner = owner               // владелец
   this.ref = ref                   // ссылка на справочник
}
```

Далее создадим записи о конкретных автомобилях на стоянке. Массив записей назовем **acar**:

```
acar = new ArrayO
acar[0] = new car("A123BX", "Иванов", aref[1])
acar[l] = new car("M345CT", "Петров", aref[l])
acar[2] = new car("E678CA",, "Сидоров", aref[0])
acar[3] = new car("K0560X", "Михайлов", aref[2])
acar[4] = new car("K895MX", "Дунаев", aref[3])
acar[5] = new car("P340HY", "Павлов", aref[2])
acar[6] = new car("03210K", "Николаев", aref[2])
```

В нашем списке всего 7 записей. Модель ВА32106 в нем встречается два раза, ГА324 — три раза, а остальные — по одному разу. Если бы мы создавали базу данных без справочника, то нам пришлось бы многократно повторять одни и те же данные при определении массива **acar**.

Далее приводятся примеры обращения к свойствам нашей базы данных:

```
acar[4].ref.model     // "ВА31111"
acar[3] . ref.name    // "Волга"
acar[3].owner         // "Михайлов"
```

Поскольку база данных представляет собой массив, с помощью методов объекта Array можно организовать фильтрацию (выборку) записей по тому или иному критерию. Попробуйте сделать это самостоятельно.

Базу данных, создание которой мы рассмотрели выше, можно отобразить на экране монитора в окне браузера. Для этого необходимо прежде всего создать HTML-программу с таким элементом, как таблица. Напомним, что таблица определяется тегами <TABLE>, <TR>, <TH>, <TD> и др. Вы можете создать в HTML-программе раздел, обрамленный тегами <SCRIPT> и </SCRIPT>, между которыми разместить рассмотренный выше код на языке JavaScript. А далее есть два пути.

- Вручную написать теги таблицы. Если количество строк в таблице невелико, то именно так и следует поступать.
- Написать программу на JavaScript, генерирующую нужную последовательность тегов и данных как строку, а затем воспользоваться методом writeQ объекта document для ее записи в текущий HTML-документ и исполнения браузером. Этот способ интереснее и предпочтительней, если таблица содержит много строк. Небольшой объем кода обеспечивается тем, что формирование строки с HTML-кодом происходит в цикле. К этому располагает то, что строки (записи) нашей базы данных определены в виде массива.

В листинге 1.1 приведен HTML-код, содержащий только сценарий. Для упрощения ситуации мы формируем таблицу только с тремя столбцами. Мы не обращали внимания на дизайн таблицы, определяемый специальными атрибутами тегов (рис. 1.14).

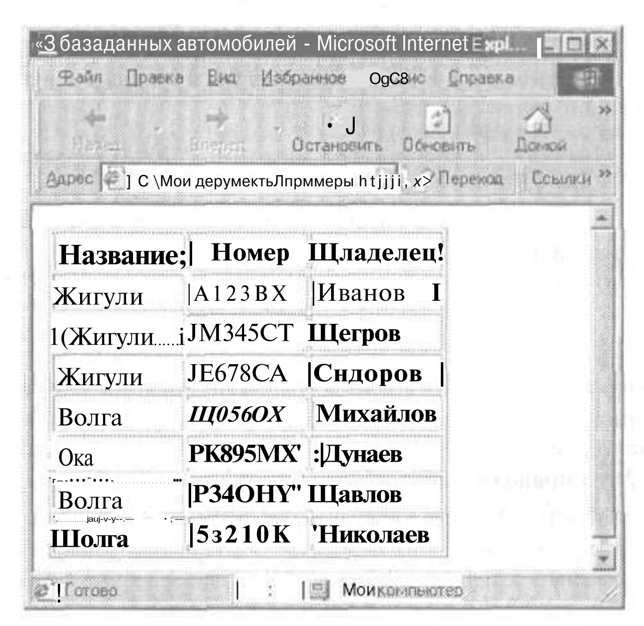


Рис. 1.14. Так выглядит база данных автомобилей в окне браузера

Листинг 1.1. Код для отображения базы данных автомобилей в браузере

```
<HTML>
<HEAO><T1T1_E>База данных автомобилей</T1T1E></HEAO>
<SCRIPT>
/* Конструкторы */
function ref(nami , model, Jrl) {          /*» конструктор справочника
                                               типов автомобилей */
this.name = name                            // название марки автомобиля
this.model = model                          // модель
this.url = url                              // URL-адрес файла с фотографией
}
function car(regnum, owner, ref) {          /* конструктор списка
                                               автомобилей на стоянке */
this.regnum = regnum                        // номер автомобиля
this.owner = owner                          // владелец
this.ref = ref                              // ссылка на справочник
}

/* Собственно база данных */

aref = new ArrayO        // массив записей справочника типов автомобилей
aref[0] = new ref("Жигули",, "Ваз2101", "pict0.gif")
aref[l] = new ref("Жигули","Ваз2106","pictl.gif")
aref [2] = new ref ("Волга","ГАЗ24", "ГА324", "pict2.gif")
aref[3] = new ref ("Ока". "ВазПП", "pict3.gif")

acar = new ArrayO
acar[0] = new car ( "A123BX", "Иванов", aref [1])
acar[1] = new car ( "M345CT" "Петров", "Петров", aref[l])
acar[2] = new car ( "E678CA", "Сидоров", aref[0])
acar[3] = new car ( "KQ56OX", "Михайлов", aref [2])
acar[4] = new car( "K895MX" "Дунаев", "Дунаев", aref[3])
acar[5] = new car ( "P340HY" "Павлов", "Павлов", aref [2])
acar[6] = new car( "03210K", "Николаев", aref[2])
strTab = "<TABLE BORDER=l> <TR>"
strTab += "<TH>Название</TH><TH> Номер</TH> <TH>Владелец</TH></TR>"
/* Формирование строк таблицы */
for(i=0;i<=acar.length-1;1++){
strTab += "<TR><TD>" + acar[i].ref.nami + "</TD><TD>" +acar[i].regnum
strTab += "</TD><TD>" + acar[i].owner + "</TD></TR>"
}

strTab += "</TABLE>"
document.write(strTab)                /* записываем строку strTab
                                         в HTML-документ и выполняем его */
</SCRIPT>
</HTML>
```

Здесь в теле оператора цикла for мы использовали разновидность оператора присвоения +=. Напомним, что выражение x += y эквивалентно выражению x = x + y. Мы дважды использовали выражение с этим оператором только лишь для того, чтобы выражение справа от него не было слишком длинным.

Мы могли бы включить в отображаемую таблицу и другие столбцы. Например, для отображения фотографий автомобилей достаточно добавить в строку strTab следующий фрагмент:

```
"<TD><IMG SRC = " + acar[i] . ref.url + '
```

1.9. Специальные операторы

В этом разделе описываются операторы, которые при программировании наJava-Script используются относительно редко. Возможно, они вам когда-нибудь потребуются. Именно поэтому я и включил в книгу данный раздел. Новички могут пока пропустить его.

1.9.1. Побитовые операторы

Побитовые (поразрядные) операторы применяются к целочисленным значениям и возвращают целочисленные значения. При их выполнении операнды предварительно приводятся к двоичной форме представления, в которой число является последовательностью из нулей и единиц длиной 32. Эти нули и единицы называются двоичными разрядами, или битами. Далее производится некоторое действие над битами, в результате которого получается новая последовательность битов. В конце концов эта последовательность битов преобразуется к обычному целому числу — результату побитового оператора. В табл. 1.4 приведен список побитовых операторов.

Таблица 1.4. Побитовые операторы

Оператор	Название	Левый операнд	Правый операнд
K	Побитовое «и»	Целое число	Целое число
|	Побитовое «или»	Целое число	Целое число
^	Побитовое исключающее «или»	Целое число	Целое число
~	Побитовое«не»	—	Целое число
<<	Смещение влево	Целое число	Количество битов, на которое производится смещение
>>	Смещение вправо	Целое число	Количество битов, на которое производится смещение
>>>	Заполнение нулями при смещении вправо	Целое число	Количество битов, на которое производится смещение

Операторы &, |,л и ~ напоминают логические операторы, но их область действия - биты, а не логические значения. Оператор - изменяет значение бита на противоположное: 0 на 1, а 1 — на 0. В табл. 1.5 поясняется, как работают операторы &, |, л.

Таблица 1.5. Работа операторов &, |,

X	Y	X&Y	X|Y	X^Y
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

Например, 2&3 равно 2, a2(3 равно 3. Действительно, в двоичном представлении 2 есть 10, а 3 — 11. Применение побитовых операторов даст в случае оператора & двоичное число 10, то есть десятеричное число 2, а в случае оператора | — двоичное число И, то есть десятеричное 3.

У операторов смещения один операнд и один параметр, указывающий, на какое количество бит следует произвести смещение. Например, 3«2 равно 12, потому что смещение влево на два бита двоичного числа 11 (десятеричное 3) дает 1100, что в десятеричной форме есть 12. Результат вычисления выражения б»2 — 1. Действительно, число 6 в двоичной форме это 110; смещение его вправо на два бита дает 1 как в двоичной, так и в десятеричной форме.

1.9.2. Объектные операторы

Оператор удаления свойств объекта (delete)

Удалить свойство объекта, а также элемент массива можно с помощью оператора **delete.** Обычно этот оператор используют для удаления элементов массива:

delete элемент

ВНИМАНИЕ

При удалении элемента массива удаляется и его индекс, но оставшиеся элементы сохраняют свои прежние индексы, а длина массива не изменяется.

Пример

```
myarray = new Array("a","b". "c", "d")
myarray.length                    // 4
delete myarray[l]
myarray.length                    // 4
myarray [0]                       // "a"
myarray[1]                        // undefined
myarray[2]                        // "c"
myarray [3]                       // "d"
```

Использование оператора delete не приводит к немедленному освобождению памяти. Решение об освобождении памяти принимается так называемым ядром JavaScript, а пользовательская программа лишь создает к этому предпосылки, но не может контролировать этот процесс абсолютно.

Оператор проверки наличия свойств (in)

Этот оператор позволяет проверить, имеется ли некоторое свойство или метод у того или иного объекта. Левый **операнд** представляет собой ссылку в виде строки на интересующее нас свойство или метод, а правый операнд — объект. Ссылка на метод содержит лишь его название без круглых скобок. Если свойство или метод содержится в объекте, то возвращается true, иначе — false. Отсюда следует, что оператор in можно применять в условных выражениях (в операторах if, switch, for, while, do-while).

Примеры

1. Например, объект document, представляющий загруженный в браузер HTML-документ, имеет метод writeQ. Чтобы убедиться в этом, следует написать выражение

```
"write" in document                    // значение равно true
```

2. Создадим объект и проверим наличие в нем некоторых свойств.

```
function Сотрудник (Имя, Отдел, Телефон, Зарплата) {
this.Имя = Имя
this.Отдел = Отдел
this.Телефон = Телефон
this.Зарплата = Зарплата
)
agent007 = new Сотрудник( "Джеймс Бонд", 5, "223-332")
"Телефон" in agent007                              // true
"Ученая степень" in agentS07                       // false
```

Оператор in поддерживается браузерами 1E5.5+,NN6+.

Оператор проверки принадлежности объекта модели (instanceof)

Этот оператор позволяет проверить, принадлежит ли некоторый объект объектной модели JavaScript. Левый операнд представляет проверяемое значение, а правый — ссылку на корневой объект, такой как Array, String, Date и т. п. Выражение с оператором instanceof возвращает true или false и, таким образом, может использоваться в условных выражениях (в операторах if, switch, for, while, do-while).

Пример

Созданный массив является экземпляром объекта Array, а последний сам является экземпляром корневого объекта Object.

```
myarray = new ArrayQ
myarray instanceof Array        // true
Array instance Obj ect          // true
Myarray instanceof String       // false
```

Оператор in поддерживается браузерами 1E5.5+, NN6+.

1.9.3. Комплексные операторы

Оператор условия (?:)

Этот оператор является сокращенной формой оператора условного перехода if... else... Его так и называют: оператор условия. Обычно он используется вместе с оператором присвоения одного из двух возможных значений, в зависимости от значения условного выражения. Синтаксис оператора условия следующий:

```
условие ? выражение! : выражение2
```

С оператором присвоения оператор условия имеет такой вид:

```
переменная = условие ? выражение! : выражение2
```

Оператор условия возвращает значение выражения выражение!, если условие истинно, в противном случае — значение выражения выражение2.

Пример

```
d = new Date()
x = d.getDateO
typedate = (x%2 == 0)&&(x > 1) ? "четное" : "нечетное"
```

Если бы в JavaScript не было оператора условия, то пришлось бы написать функцию, которая работала бы как оператор условного перехода if, но, в отличие от него, возвращала значение. Вот пример этой функции:

```
function iff(condition, exprl, expr2){
if (condition)
   return eval(exprl)
else
   return eval(expr2)
}
```

Для данной функции нельзя было выбрать название if, поскольку это ключевое слово. Потому было взято наиболее близкое к нему. Обратите внимание: предполагается, что выражения передаются функции как строки, содержащие выражения. Однако допустимы значения и других типов.

Оператор определения типа (tipeof)

Этот оператор используется для проверки, относится ли значение к одному из следующих типов: string, number, boolean, object, function или undefined. Значение, возвращаемое оператором typeof, является строковым. Оно содержит одно из перечисленных выше названий типа. Единственный операнд пишется справа от ключевого слова typeof.

Примеры

```
var x
n = 3.14
N = new Number(3.14)
s = "Привет всем!"
a = new Array(l, 2, 3, 4, 5)
function f(){}

typeof x     // "undefined"
typeof n     ' "number"
typeof N     // "object"
typeof s     // "string"
typeof a     // "object"
typeof f     // "function"
```

1.10. Приоритеты операторов

Выше мы уже говорили, что в выражениях операторы выполняются в порядке, определяемом с помощью круглых скобок, согласно приоритетам; операторы с одинаковыми приоритетами выполняются слева направо. В этом разделе мы уточним приоритеты всех операторов (табл. 1.6).

Круглые скобки, с помощью которых можно установить любой порядок выполнения операторов в выражении, можно считать операторами. Они обладают наивысшим приоритетом. Эти скобки могут образовывать структуру вложенности. Выражения, заключенные во внутренние круглые скобки, выполняются раньше тех, которые заключены во внешние круглые скобки. Интерпретатор JavaScript начинает анализ выражения именно с выяснения структуры вложенности пар круглых скобок.

На втором месте по приоритету находится вычисление индексов массивов и определения самих элементов массивов. Индексы массивов, как известно, заключены в квадратные скобки.

На последнем месте находится запятая, как разделитель параметров.

Таблица 1.6. Распределение операторов по приоритетам

Приоритет	**Оператор**	**Комментарий**
1	()	От внутренних к внешним
	[ ]	Значение индекса массива
	function()	Вызов функции
2	!	Логическое «не»
	~	Побитовое «не»
	-	Отрицание
	++	Инкремент (приращение)
	--	Декремент
	new	
	typeof	
	void	
	delete	Удаление объектного элемента
3	*	Умножение
	/	Деление
	%	Деление по модулю (остаток от деления)
4	+	Сложение (конкатенация)
	-	Вычитание
5	<<	Побитовые сдвиги
	>>	
	>>>	
6	<	Меньше
	<=	Не больше (меньше или равно)
	>	Больше
	>=	Не меньше (больше или равно)
7	==	Равенство
	!=	Неравенство
8	&	Побитовое «и»
9	^	Побитовое исключающее «или»
10	|	Побитовое «или» (дизъюнкция)
11	&&	Логическое «и» (конъюнкция)
12	||	Логическое «или»

продолжение ⇨

Таблица 1.6 *(продолжение)*

Приоритет	Оператор	Комментарий
13	?	Условное выражение (оператор условия)
14	=	Операторы присвоения
	+=	
	-=	
	*=	
	/=	
	%=	
	« =	
	>=	
	>>=	
	&=	
	^=	
	|=	
15	,	Запятая (разделитель параметров)

Кроме приоритетов следует также учитывать, что сложные логические выражения, состоящие из нескольких более простых, соединенных операторами И и ИЛИ, выполняются по так называемому принципу короткой обработки. Это означает, что значение всего выражения бывает можно определить, вычислив лишь одно или несколько более простых выражений, не вычисляя остальные. Например, выражение x&&y вычисляется слева направо; если значение x оказалось равным false, то значение y не вычисляется, поскольку и так известно, что значение всего выражения равно false. Аналогично, если в выражении x || y значение x равно true, то значение y не вычисляется, поскольку уже ясно, что все выражение равно true.

ВНИМАНИЕ

Если вы хотите проверить правильность сложного логического выражения, то необходимо проверить отдельно все его составные части. Если какая-нибудь составная часть выражения содержит ошибку, то она может остаться невыявленной, поскольку эта часть выражения просто не выполнялась при тестировании.

1.11. Зарезервированные ключевые слова

Ключевые слова JavaScript, применяемые для обозначения элементов синтаксиса языка, а также другие, зарезервированные на будущее, нельзя использовать в качестве имен переменных, функций и объектов (табл. 1.7). В крайнем случае можете просто несколько модифицировать ключевое слово, изменив регистр, добавив какой-нибудь префикс (например, символ подчеркивания) или суффикс.

Слово interface является ключевым — и, следовательно, его нельзя использовать в качестве имени пользовательского объекта или функции. Однако можно применить некоторую модификацию этого слова: myinterface, xlnterfaceи т. п.

ВНИМАНИЕ

Использование символов в различном регистре часто приводит к недоразумениям, обусловленным тем, что одна и та же по вашему замыслу переменная из-за небрежности встречается в программе в различном регистре. Поскольку JavaScript является регистрозависимым языком, он воспринимает эти переменные как различные.

Если вы хотите использовать различный регистр в именах, то придерживайтесь определенных правил. Например, только первый символ имени всегда является прописной буквой.

СОВЕТ

Следует также иметь в виду, что со временем список зарезервированных ключевых слов может расширяться. Чтобы ваши сценарии не перестали работать в будущем, подбирайте имена для своих объектов из специфических (не общеупотребительных) слов.

Таблица 1.7. Список ключевых слов

abstract	else	int	super
boolean	extends	interface	switch
break	false	long	synchronized
byte	final	native	this
case	finally	new	throw
catch	float	null	throws
char	for	package	transient
class	function	private	true
const	goto	protected	try
continue	if	public	typeof
default	Implements	reset	var
delete	import	return	void
do	in	short	while
double	instanceof	static	with

Глава 2. Основы создания сценариев

В этой главе мы начнем с краткой истории программирования, а затем рассмотрим основные понятия, объекты JavaScript и приемы работы с ними. Их нужно освоить, прежде чем заниматься более узкими задачами. В главе 4 вы познакомитесь с конкретными примерами сценариев, но чтобы научиться решать задачи, выходящие за рамки конкретных примеров, прочитайте данную главу. В этом случае вы с легкостью и с интересом усвоите справочный материал, изложенный в главе 3.

2.1. Из истории программирования

Давным-давно, примерно с 1950 по **1960** год, в программировании не было сколько-нибудь явно выраженных технологий. Не существовало понятия вызова функции, данные хранились где попало, для управления вычислительным процессом широко использовались операторы перехода, в том числе и оператор GO TO (перейти туда, кудая сказал). Программные коды были огромными, содержали массу ошибок и стоили очень дорого. Операторы GO TO бросали читателя исходных текстов то вверх, то вниз по листингу исходного текста программы. Отладка программ занимала львиную долю всего времени разработки. Тогда программировали в цифровых кодах команд, а переход к языку мнемокодов Assembler казался этапом значительной автоматизации работы программистов. Нормой производительности труда программистов было 3-5 команд в день. В действительности программист за месяц писал тысячи кодов, хотя результат состоял всего лишь из нескольких их десятков. А зачем он это делал, если ему платили по норме? В те романтические **годы** программистом мог быть только специалист по вычислительной технике. При этом программисты, как правило, были еще и математиками. Математиками их называли до середины 80-х годов прошлого века независимо от того, имели они специальное образование или нет, понимали ли они хоть что-нибудь в этой изящной и прекрасной науке или нет.

В период с 1960 по 1975 год появились первые функциональные языки программирования, такие как FORTRAN и ALGOL. Эти языки в основном предназначались для поддержки формульных вычислений, необходимых ученым и инженерам. Расчет траекторий орбит космических аппаратов и планет, создание программ наведения антенн, прогноз погоды и параметров электронных схем — далеко не

полный перечень задач, обслуживаемых программными языками данного типа. Эти языки позволяли создавать функции, что ускорило создание более объемных программ. Однако операторы GO TO по-прежнему изобиловали в программах, их отладка, как и раньше, оставалась трудоемким и длительным делом, а мысль о необходимости структуризации данных и собственно программ только зарождалась. В это же время возникли первые операционные системы.

В конце 70-х годов стало ясно, что программы и данные должны иметь четко выраженную структуру, чтобы их можно было быстро разрабатывать, отлаживать, модифицировать и интегрировать с другими программными комплексами. На эту тему написаны сотни книг и статей. Тема публикаций и дискуссий — структурное программирование. Оно в итоге значительно улучшило качество программных продуктов. PL/I и С — наиболее типичные и знаменитые языки структурного программирования. Появилась возможность создавать сложные структуры данных с помощью средств языка. Например, программист мог создать структуру Автомобиль для хранения в ней всехнеобходимыхданныхоб автомобиле: марки, номера, цвета, имени владельца и т. п. При этом все данные хранились в одном месте, а не рассеивались по множеству переменных. Те, кто знаком с программированием баз данных, могут удивиться: а что тут особенного? Одно дело сохранять данные в соответствии с некоторой структурой на диске, а другое — обеспечивать поддержку структуры на уровне языка.

С 1990 года началась эра так называемого объектно-ориентированного программирования (ООП). По существу, ООП основывается на расширении концепции структуры. Если раньше в структуре хранились лишь статические данные, то теперь стало возможным хранить и описания активных элементов, называемых методами. Такие структуры в языках ООП называются классами, или объектами. Методы объекта описывают его поведение. Например, разрабатывая компьютерную игру, программист может создавать объекты для всех ее персонажей. Кроме статических характеристик (имя, внешний вид, координаты положения), объект может содержать и описания методов изменения положения, своего внешнего вида и реакции на действия игрока. Другими словами, объект представляет собой некий контейнер, содержащий переменные и функции, которые применительно к объекту называются его свойствами и методами. Замечательно еще и то, что в языках, поддерживающих ООП, из одних объектов можно делать другие, добавляя к ним новые свойства. При этом созданные таким образом новые объекты наследуют свойства исходных (родительских) объектов. Например, объект Студент наследует свойства объекта Человек, а последний наследует свойства объекта Млекопитающее. Благодарямеханизмунаследованияпрограммист освобождается от необходимости повторять историю создания новых объектов начиная от Адама.

Самый известный и широко используемый язык ООП — C++. Язык С (просто Си) — мощный, но низкоуровневый язык программирования. В нем есть все необходимое для создания объектов, однако ответственность за это полностью ложится на программиста. Язык C++, напротив, большую часть рутинной работы по созданию объектов (классов) берет на себя. Существуют и другие объектно-ориентированные языки (например, Java и Object Pascal), но C++ стал стандартом ООП-языка де-факто. Если какой-либо язык так или иначе поддерживает то

же, что и C++, то говорят, что это ООП-язык. Однако не все языки, поддерживающие работу с объектами, являются ООП-языками в полной мере. Это, по существу, означает, что они поддерживают не все механизмы работы с объектами, которые имеются в C++. Так, например, JavaScript не вполне ООП-язык, поскольку его последняя версия 1.5 не поддерживает концепцию наследования свойств объектов в том объеме, в каком это реализовано в C++. Заметим при этом, что следующая версия JavaScript, как ожидается, уже будет полностью соответствовать концепции ООП.

Не для всех программистов, воспитанных на идеях функционального и структурного программирования, переход к ООП был гладким и естественным. В начале 90-х годов появляется множество популярных книг, авторы которых пытались с помощью простых аналогий из повседневной жизни объяснить суть объектов. Обычно в качестве примеров приводились магнитофоны и микроволновые печи с кнопками и регуляторами, которые должны были проиллюстрировать идею свойств и методов. Признаюсь, мне тоже было нелегко освоить новую философию программирования. Вместе с тем я видел, как новички без особых затруднений осваивали C++ и другие объектные языки. Они попросту не были обременены стереотипами. В конце концов и я с этим разобрался. Во-первых, в отношении ООП я задавал неправильный вопрос «а зачем это надо?», который только сбивал с толку. Дело в том, что я уже давно мыслил в терминах ООП и даже создавал объекты, но посредством функциональныхязыков. Поэтому, естественно, я не мог получить хороший ответ на свой вопрос «зачем?». Для меня тогда единственным оправданием ООП-языков оставалось лишь наличие компиляторов для них, осуществляющих проверку синтаксической правильности, и средства отладки программ. Во-вторых, аналогия объектов с бытовыми приборами вызывала у меня неверные ассоциации, скорее удаляя от понимания, чем приближая к сути. Возможно, это связано с индивидуальными особенностями моего восприятия. Тем не менее лучшей, на мой взгляд, интерпретацией объекта является контейнер, содержащий переменные и функции. Представьте себе, что я написал некую программу, предназначенную для решения некоторых задач. Свое творение я назвал объектом и сделал его доступным для пользователя или программиста. Это означает, что я опубликовал описание этого объекта, в котором указал, какие внутренние переменные и функции могут быть использованы и каким образом, чтобы объект работал по своему назначению. Иначе говоря, я просто вывел наружу датчики и органы управления своим объектом, а все остальное скрыл от глаз пользователя. Со временем я могу усовершенствовать программный код своего объекта, не сообщая об этом пользователю, и, что еще важнее, при соблюдении определенных условий я могу не тестировать свой модифицированный объект в комплексе с другими.

В связи с появлением многозадачных операционных систем, таких как Windows, OS/2, Unix и Linux, открылись возможности для так называемого событийного программирования. Программные единицы (объекты, в частности) могут реагировать на действия пользователя (например, щелчок кнопкой мыши, нажатие на клавишу и др.), а также передавать сообщения о событиях другим программным единицам. При этом внешне ситуация выглядит так, будто одновременно функционирует множество объектов, в какой-то степени автономных, но действующих

согласованно, взаимообусловлено. С одной стороны, это открывает огромные возможности довольно легко создавать интересные и сложные программы. С другой стороны, при необдуманных решениях могут возникнуть непредвиденные и плохо контролируемые ситуации.

Язык JavaScript создан, главным образом, для разработки интерактивных приложений, то есть программ, реагирующих на действия пользователя. Действия пользователя инициируют события, которые и обрабатываются в сценариях, написанных на JavaScript. Вместе с тем на этом языке можно писать и другие программы, например создавать игры, информационно-справочные системы и т. п. Браузеры, выполняющие программы на JavaScript, предоставляют программисту огромное множество уже готовых объектов. Таким образом, вы освобождаетесь от рутинной части работы по созданию графической среды и элементов управления, вам остается только использовать их в соответствии с вашими задачами. В большинстве случаев программы на JavaScript оказываются достаточно короткими, поэтому их может разработать и отладить даже новичок. Отметим еще одно важное обстоятельство: JavaScript используется, как правило, совместно с другим языком — HTML. С одной стороны, это требует от программиста знания двух языков, а с другой — существенно упрощает задачу организации ввода и вывода информации. Наконец, именно благодаря HTML, который всего за несколько лет освоили миллионы, JavaScript стал чрезвычайно популярным языком среди обычных пользователей компьютеров, а не только программистов-профессионалов.

Те, кто знаком с языком С, легко освоят JavaScript, поскольку многие языковые конструкции им уже известны. Они должны лишь привыкнуть к свободе обращения с типами данных, которую предоставляет JavaScript, а также освоить концепцию объектной модели документа.

Несмотря на похожие названия, между JavaScript и Java очень мало общего. Java похож на C++, но, в отличие от него, является не зависимым от платформы среды выполнения.

2.2. От простого до динамического HTML

2.2.1. Простой HTML

Для создания веб-страниц разработан довольно простой язык HTML (Hyper Text Markup Language — язык разметки гипертекста). Изначально он задумывался как язык разметки документа, содержащего текстовую и графическую информацию вместе с элементами управления, такими как ссылки на другие документы. Тексты, содержащие элементы, которые являются ссылками на другие документы, называются гипертекстами, а сами ссылки называют еще гиперссылками. Кроме текстовой и графической информации, а также ссылок в документ можно включать аудио- и видеофрагменты. Чтобы создать такой документ, достаточно в обычном текстовом редакторе (например, в Блокноте Windows) написать программу на языке HTML Такие программы сохраняются в файлах с расширением htm или html, а выполняются они в веб-браузере, например Microsoft Internet Explorer или Netscape Navigator.

Инструкции HTML называют тегами, или дескрипторами. Они выделяются угловыми скобками (< и >). Каждый тег имеет свое название, являющееся ключе-

вым словом. Спецификация тега обеспечивается его параметрами, которые обычно называют атрибутами. Атрибуты могут иметь значения (аргументы). Названия тегов и атрибуты можно писать в любом регистре. Атрибуты, если они имеются, пишутся за названием тега в произвольном порядке через пробел. Формат тега имеет следующий вид:

<НАЗВАНИЕ_ТЕГА [АТРИБУТ1 [= значение!] . . . [АТРИБУТМ [= значением]]>

Квадратные скобки показывают, что заключенные в них элементы не обязательны . Существуют теги без атрибутов (например, тег перехода на новую строку < BR>), а также атрибуты без аргументов (например, NORESIZE в теге <FRAME>). Даже если для тега предусмотрены атрибуты, во многих случаях их можно не указывать, используя значение по умолчанию. Так, тег <IMG SRC = "picture.jpg"> дает браузеру команду вывести на экран графический объект из файла picture.jpg. Здесь ключевое слово IMG — название тега, SRC — атрибут, а "picture.jpg" — аргумент (значение). Атрибут ID присущ любому тегу, но если он не используется, то по умолчанию принимается, что его значение — пустая строка.

Теги можно писать в одной строке или в нескольких, делая переносы в любом месте. Невидимый служебный символ перевода строки и возврата каретки, вставляемый в редакторе при нажатии клавиши Enter, не воспринимается браузером как разделитель тегов. Можно считать, что браузер воспринимает последовательность тегов как единую строку символов.

Большинство тегов являются контейнерными. Это означает следующее.

- Тегу<ТЕГ. .> обязательно соответствует заключительный тег</ТЕГ>.
- Между этими тегами можно разместить другие теги, контейнерные или нет. В связи с этим можно говорить о вложенности одних тегов в другие.

Например, теги тела документа <BODY>, ссылки <A . . .>, раздела <DIV>, таблицы <TABLE> и др. являются контейнерными, то есть им соответствуют заключительные теги </BODY>, </A>, </DIV>, </TABLE>. Говоря «внутри тега <ТЕГ>», мы имеем в виду то, что расположено между тегами <ТЕГ> и </ТЕГ>. В приведенном ниже примере показано, что внутри тега ссылки <A> можно разместить тег графического изображения <IMG>, создав таким образом графическую ссылку:

```
<A HREF = "www.yandex.ru"><IMG SRC ="banner.gif"></A>
```

HTML-программа (HTML-код), формирующая документ, начинается тегом < HTML> и заканчивается тегом </HTMl>. Таким образом, <HTML> является контейнерным тегом, причем самого верхнего уровня.

В HTML-коде существуют теги, которые никак не проявляются в окне браузера какими бы то ни было визуальными эффектами (например, <HEAD>, <META>, <SCRIPT>). Вместе с тем существует множество тегов, которым соответствуют определенные видимые элементы документа (веб-страницы). Например, тегу <IMG> соответствует изображение, Тегу <BUTTON> — кнопка, тегу<INPUT> — поле ввода данных, переключатель или кнопка, в зависимости от значения атрибута TYPE.

Приведенный ниже HTML-код формирует простую веб-страницу, содержащую заголовок первого уровня (<H1>), изображение (<IMG>), ссылку (<A>) и форму (<FORM>), которая содержит поле ввода данных (<INPUT>) и кнопку (<BUTTON>).

```
<HTML>
<H1>Моя веб-страница</H1>
<IMG SRC="pict.'pg"> ,
<A HREF = "www. admi ral. ru/~dunaev">Caiu автора</a>
<FORM>
<INPUT TYPE="text" VALUE="">
<p>
<ВиТТОМ>Нажми здесь</В1ЯТОМ>
</FORM>
</HTML>
```

На рис. 2.1 приведен этот HTML-документ в окне браузера, а также показано соответствие его элементов тегам HTML-кода.

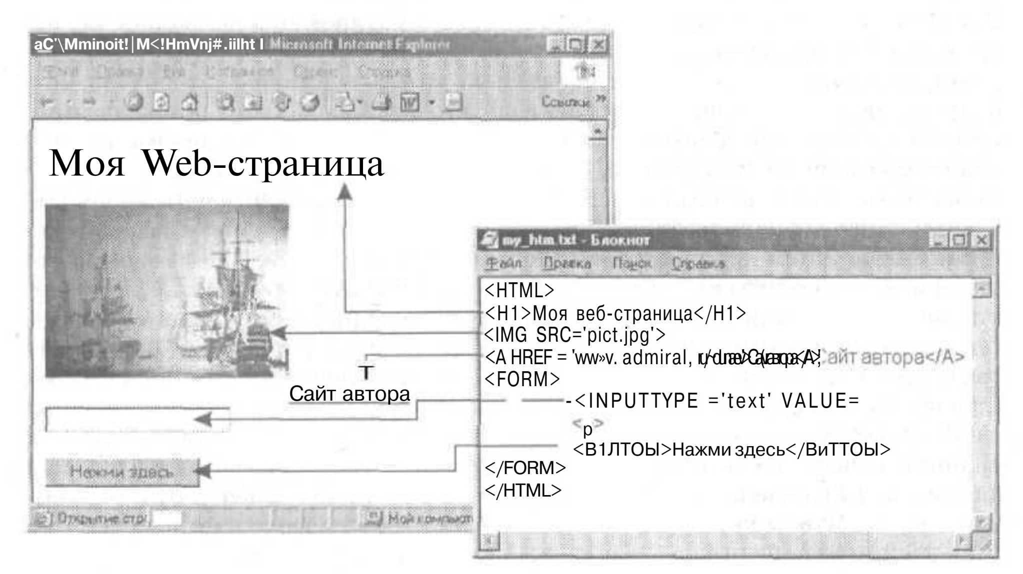


Рис. 2.1. Тегам HTML соответствуют элементы HTML-документа в окне браузера

С помощью специальных тегов можно форматировать текстовые данные: управлять шрифтом, задавать абзацы и способ выравнивания, в том числе взаимное расположение текста и изображений.

Если не использовать специальных средств позиционирования, то элементы в документе будут располагаться в порядке их появления в HTML-коде соответствующих им тегов. При этом если не использовать теги перевода строки (
) или абзаца (<P>), то элементы будут располагаться рядом друг с другом по горизонтали или по вертикали, в зависимости от размеров окна браузера и их собственных размеров. Теги
 и <P> являются довольно слабыми средствами форматирования.

Один из часто применяемых способов разметки документа основан на использовании группы тегов, определяющих таблицу (<TABLE>, <TR>, <TD> и др.). Таблицы можно создавать не обязательно с видимыми границами (рамками). С помощью таблицы можно просто разбить все поле документа на прямоугольные ячейки одинаковых или различныхразмеров и размещать в них нужные элементы (тексты, изображения, кнопки, ссылки и т. п.). Во многих случаях этого способа разметки вполне достаточно.

2.2.2. Динамический HTML

Выше мы бегло рассмотрели то, что называют простым, или классическим, HTML. Вскоре после его появления выяснилось, что он не вполне отвечает потребностям разработчиков и пользователей веб-сайтов. Во-первых, нужны были более мощные и гибкие средства форматирования. Во-вторых, требовалась возможность вставки в HTML-документ объектов сторонних производителей. В-третьих, требовались средства для управления содержимым HTML-документа, загруженного в браузер, а также для обработки действий пользователя (манипуляции мышью, нажатия на клавиши). Иначе говоря, было необходимо, чтобы HTML-документ являлся динамическим и интерактивным (мог взаимодействовать с пользователем).

В ответ на потребность в мощных средствах форматирования появились так называемые CSS (Cascading Style Sheets — каскадные таблицы стилей). Теперь параметры тегов стало возможным задавать не только с помощью атрибутов, но и в строке, являющейся значением специального атрибута STYLE. С помощью этого атрибута можно определить индивидуальные параметры форматирования и позиционирования практически для всех тегов. Кроме атрибута STYLE был введен в обращение контейнерный тег <STYLE>. В теге <STYLE> можно задать индивидуальные стили для ряда тегов, а также создать произвольные стили и закрепить за ними имена. Затем для придания какому-нибудь тегу свойств поименованного стиля можно обратиться к нему с помощью атрибута CLASS = имя_стиля. С помощью стилей можно определить параметры форматирования текстов, задать фильтры (визуальные эффекты) для текстовых и графических элементов, а также задать три координаты позиционирования элементов. Позиционирование, намойвзгляд,— главная функциональная возможность CSS. Координаты top и left, задаваемые с помощью стиля, определяют положение элемента на плоскости документа, а координатаz-indexуказывает еще и слой документа. Так, задавая значения top и left для различных элементов, можно добиться того, что одни элементы будут накрывать другие. Чтобы указать, какой из них должен быть выше или ниже другого (независимо от порядка следования в HTML-коде), служит параметр z-index.

Объекты сторонних производителей, созданных с помощью различных языков программирования, вставляются в HTML-документ посредством контейнерных тегов <OBJECT> и <EMBED>. С их помощью можно встроить в HTML-документ звук, видео, Flash-анимацию, векторную графику, таблицы базы данных и многое другое.

Для управления элементами HTML-документов и даже самим браузером, генерации новых документов, организации диалогового взаимодействия с пользователем, выполнения каких-то расчетов и обработки данных в HTML была предусмотрена интеграция со специальными языками программирования. Программы, написанные на этих языках, называют сценариями (scripts). Стандартным языком сценариев является JavaScript. Его должны уметь интерпретировать все веб-браузеры. Браузер Microsoft Internet Explorer помимо JavaScript воспринимает еще один язык — VisualBasicScript, чего нельзя сказать о Netscape Navigator.

Каким образом JavaScript взаимодействует с HTML-документом и браузером? Дело в том, что браузер и загруженный в него HTML-документ представлены внутри браузера посредством иерархического множества объектов — так называемой объектной модели. Для сценария на JavaScript объекты браузера и HTML-документа образуют доступную среду. Что такое объекты и как с ними обращаться, мы уже узнали из главы 1. Чтобы использовать объекты браузера и документа,

загруженного в него, необходимо знать их названия, свойства и методы. Но об этом немного позже.

2.3. Где, что и как делают сценарии

2.3.1. Расположение сценариев

Программы, работающие с объектами HTML-документа, называют сценариями. Вы можете написать программу на языке JavaScript. Она может иметь самостоятельную ценность, используя собственные ресурсы JavaScript, но, кроме Того, она способна взаимодействовать с объектами среды, окружающей интерпретатор языка. Интерпретатор JavaScript, встроенный в веб-браузер, предоставляет пользователю возможность использовать средства языка для доступа к ресурсам браузера и всего того, что в нем находится в данный момент. А именно к свойствам браузера и документа, загруженного в него. Конечно, вам не терпится узнать, как добраться до этих ресурсов, чтобы воздействовать на них. Это делается с помощью сценариев, написанных на языке JavaScript в некоем месте. А где именно? Есть несколько вариантов размещения программ на JavaScript, выполняющих роль сценариев для HTML-документов. Ниже мы их и рассмотрим.

Сценарии можно писать непосредственно в HTML-документе, а также в отдельных текстовых файлах, которые вызываются из HTML-документа. Проще всего размещать сценарий непосредственно в HTML-документе, чаще всего так и поступают. Сразу заметим, что размещение сценария в отдельном файле совсем не сложно, но не всегда оправданно из экономических соображений. Необходимо накопить достаточное количество программных заготовок, чтобы решиться использовать их в последующих проектах в качестве библиотек. Новички могут пока об этом не задумываться, достаточно помнить, что рано или поздно такая задача возникнет. Итак, сейчас мы займемся размещением сценария в HTML-документе, чтобы иметь возможность использовать их для оперативной модификации HTML-документа, а следовательно и веб-страницы.

Сценарий в HTML-документе можно разместить несколькими способами. Свобода подкупает, требуя взамен принять хоть какое-нибудь решение. Трудности наступают именно при принятии волевого решения. Мы начинаем раздумывать о правильности сделанного выбора. Именно тогда и возникают мечты о приемлемом рецепте. Рецепты можно выдавать только при четко поставленном диагнозе и фиксированном анамнезе. В программировании это трудно сделать. Программирование — это искусство, хотя оно и близко к науке. Первый принцип искусства программирования — знаменитая «бритва Оккама»: не умножай сущностей. Иначе говоря — не усложняй. Все, заканчиваем лирику и приступаем к делу.

Стандартным является размещение сценария в контейнерном теге <SCRIPT>, то есть между тегами <SCRIPT> и </SCRIPT>. Встречая тег <SCRIPT>, браузер «понимает», что за ним начинается код сценария. Заключительный тег </SCRIPT> указывает браузеру, что код сценария закончился. Все, что находится вне этих тегов, браузер воспринимает как HTML-код. Контейнер <SCRIPT> может располагаться в любом месте HTML-документа и даже не один раз. От его расположения иногда может зависеть функционирование всего HTML-документа, но об этом мы скажем ниже.

Контейнерный тег <SCRIPT>, объемлющий код сценария, может содержать следующие атрибуты.

- LANGUAGE — язык сценария; возможные значения:
  - "JavaScript", "JScript";
  - "VBScript", "VBS".

  Если атрибут LANGUAGE не указан, то в Internet Explorer подразумевается JScript.
- SRC — указывает файл (имя или URL-адрес), содержащий код сценария. Этот атрибут используется в том случае, если сценарий расположен не в HTML-документе, а в отдельном файле.

Примеры

```
<SCRIPT LANGUAGE="JavaScript">
. . .                                   // код сценария
</SCRIPT>
<SCRIPT LANGUAGE="JScript" SRC = "myscripts.js"></SCRIPT>
```

Современные браузеры распознают еще и версию языка. Например, Internet Explorer 4.0 и Netscape Navigator 4.0 знают версию языка JavaScript 1.2. Во время написания этой книги была доступна версия JavaScript1.5, о которой знают Internet Explorer 6.0 и Netscape Navigator 6.0. Если версия языка JavaScript в значении атрибута LANGUAGE указана, а браузер не знает ее, то сценарий будет им просто проигнорирован. Если же версия языка не указана, то браузер, которому она не знакома, может неправильно выполнять некоторые выражения сценария или даже выдавать сообщения об ошибках.

Редакция языка JavaScript для Internet Explorer называется JScript. Вместе с тем в Internet Explorer можно использовать и LANGUAGE = "JavaScript". В браузере Netscape Navigator значимым является только LANGUAGE = "JavaScript", ссылка LANGUAGE = "JScript" будет им проигнорирована. Поэтому при разработке сценариев, рассчитанных для различных браузеров, рекомендуется использовать ссылку LANGUAGE = "JavaScript". В примерах, приводимых в настоящей книге, мы не будем указывать язык, на котором написан сценарий (из соображений экономии места).

Старые браузеры, появившиеся раньше JavaScript, игнорируют теги <SCRIPT> и </SCRIPT>, а все, что находится между ними, интерпретируют как содержимое HTML-документа. Результат может быть самым неожиданным. Чтобы уменьшить вероятность отображения кода сценария в окне старого браузера, следует заключить его в дескрипторы комментария <!— и -->. Новые браузеры, поддерживающие сценарии, будут игнорировать эти символы, выполняя код сценария, а старые (не понимающие сценарии), наоборот, будут игнорировать код сценария. Вот как используются символы комментария предохранения сценария от старых браузеров:

```
<SCRIPT LANGUAGE="JavaScript" >
<!--
                                        // код сценария

</SCRIPT>
```

Обратите внимание на заключительные символы тега комментария, перед которыми стоят две косые черты. Без них JavaScript будет пытаться интерпретировать символы --> как заключительные символы комментария HTML, а с ними он просто их проигнорирует. В примерах, приводимых в настоящей книге, мы не будем указывать символы комментария для адаптации к старым браузерам из сообра-

жений экономии времени и места. Однако помните, что современная культура оформления сценариев обязывает нас делать это в своих практических приложениях.

Если сценарий располагается в отдельном файле, то в нем, разумеется, теги <SCRIPT> и </SCRIPT> не пишутся. Как уже отмечалось, файлы со сценариями на JavaScript являются обычными текстовыми файлами. В принципе, они могут иметь любое расширение имени, но, как правило, используется расширение js. В отдельных файлах обычно размещают библиотеки функций (определения функций), а также сценарии, использующиеся в нескольких HTML-документах одного или нескольких сайтов. Сценарий, загруженный из внешнего файла, можно представить себе просто как его вставку в HTML-документ. Вот пример:

```
<HTML>
. . .
<SCRIPT>
function myfunc() {
. . .
}
</SCRIPT>
<SCRIPT SRC = "[tiylibraryl. js"></SCRIPT>
<SCRIPT SRC = "myprogram. js"></SCRIPT>
. . .
</HTML>
```

Здесь в HTML-документе расположены три раздела (секции) сценария, два из которых загружаются из отдельных файлов. В первом разделе сценария дано определение некоторой функции.

Сценарий можно также писать как строку операторов, разделенных точкой с запятой, взятую в кавычки. В такой форме сценарии обычно используются в качестве обработчиков событий. Об этом более подробно будет рассказано в подразделе 2.3.2.

Вызовы функций и их определения могут располагаться в произвольном порядке, только если они расположены в одном и том же контейнере <SCRIPT>. Если вы располагаете их в разных контейнерах <SCRIPT>, то необходимо, чтобы определение функции располагалось выше ее вызова. Аналогично, если определения функций находятся в отдельном файле, то его необходимо загрузить в HTML-документ раньше вызовов этих функций. Ниже приведены примеры правильного и неправильного расположения сценариев в HTML-документе:

Правильно	Неправильно
<HTML>	<HTML>
. . .	. . .
<SCRIPT>	<SCRIPT>
function myfunc(){	. . .
} . .	myfuncO
</SCRIPT>	</SCRIPT>
<SCRIPT>	<SCRIPT>
. . .	function myfunc(){
myfuncO	. . .
	}
</SCRIPT>	</SCRIPT>
</HTML>	</HTML>

При попытке загрузить и выполнить неправильный вариант HTML-кода появится диалоговое окно с сообщением об ошибке «Ошибка: Предполагается наличие объекта». Браузер интерпретирует HTML-теги последовательно. Так, встретив выражение вызова функции myfunc() в одном контейнере <SCRIPT>, браузер еще не имеет в памяти определения этого объекта (функции), расположенного в другом контейнере <SCRIPT>. Если же определение функции и ее вызов находятся в одном и том же контейнере <SCRIPT>, то его содержимое сначала загружается в память, а затем анализируется. Если интерпретатор встречает вызов функции, то он ищет ее определение в памяти и при удачном результате выполняет код этой функции.

Следующие два варианта являются правильными, потому что определение функции и ее вызов находятся в одной секции сценария (в одном контейнере <SCRIPT>):

Правильно	Правильно
<HTML>	<HTML>
. . .	. . .
<SCRIPT>	<SCRIPT>
. . .	. . .
function myfunc(){	myfunc()
. . .	. . .
}	function myfunc(){
. . .	. . .
myfunc()	}
. . .	. . .
</SCRIPT>	</SCRIPT>
. . .	. . .
</HTML>	</HTML>

Если необходимо, чтобы сценарий загрузился в браузер прежде, чем загрузятся элементы HTML-документа, то его следует расположить в верхней части HTML-кода. В этом случае сценарий обычно располагают в контейнере <HEAD> (в заголовке документа). Это самое лучшее место для расположения функций.

Если требуется, чтобы сценарий загрузился после загрузки всех элементов HTML-документа, то возможны следующие два варианта. Во-первых, можно расположить сценарий в конце HTML-документа. Во-вторых, можно использовать атрибут-событие onload в контейнерном теге <BODY>, который задает основную часть HTML-документа. В последнем случае значением атрибута onload обычно является строка, содержащая имя функции. Определение этой функции, как правило, содержится в контейнере <SCRIPT>, размещенном в контейнере заголовка HTML-документа <HEAD>. Обратите внимание, что при использовании атрибута-события onload в теге <BODY> обработчик этого события выполняется по завершении загрузки всех элементов, определенных в контейнере <BODY>, а не в процессе их загрузки. Вот пример:

```
<HTML>
<HEAD>
<SCRIPT>
function myfunc () {
. . .
}
</SCRIPT>
</HEAD>
. . .
<BODY onload = "myfunc()">
теги
```

```
</BODY>
. . .
</HTML>
```

2.3.2. Обработка событий

Одним из главных (но далеко не единственным) назначений сценариев в HTML-документе является обработка событий, таких как щелчок кнопкой мыши на элементе документа, помещение указателя мыши на элемент, перемещение указателя с элемента, нажатие клавиши и т. п. Большинство тегов HTML имеют специальные атрибуты, определяющие события, на которые могут отреагировать соответствующие элементы. Список всех допустимых событий довольно обширен и рассчитан практически на все случаи жизни. События называются довольно просто, особенно если вы знаете английский язык. Например, щелчок левой кнопкой мыши — onclick; изменение в поле вводаданных — onchange; событие onmouseoverпроисходит, когда указатель мыши помещается на элемент HTML-документа. Список событий мы рассмотрим позже. Значением таких атрибутов-событий в тегах HTML является строка, содержащая сценарий, выполняющий роль обработчика события. Например, следующий HTML-код определяет заголовок второго уровня, который реагирует на щелчок кнопкой мыши тем, что выполняет некоторую функцию myfuncQ:

```
<H2 onclick = "myfunc( )">Щелкни здесь</H2>
```

Для одного и того же элемента можно определить несколько событий, на которые он будет реагировать. Другими словами, для одного и того же тега можно указать несколько атрибутов-событий. Имена этих атрибутов, как и других, можно писать в любом регистре. Порядок следования атрибутов не имеет значения.

Итак, значением атрибута-события, как уже отмечалось, является код сценария, заключенный в кавычки. Этот сценарий называют также обработчиком события. В приведенном выше примере обработчиком события onclick является функция myfuncQ. Если обработчик события содержит несколько выражений, то они разделяются точкой с запятой.

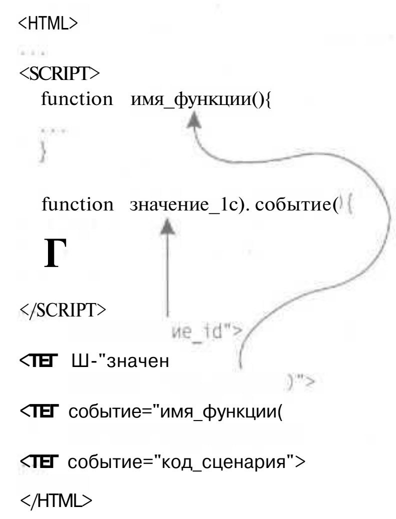


Рис. 2.2. Различные варианты оформления обработчиков событий

Пример

<H2 onclick = "v an x = 5; x = x + myfunc()">Щелкни здесь</H2>

Обычно обработчики событий оформляются в виде функций, определения которых помещают в контейнерный тег <SCRIPT>.

В качестве примера ниже приведены два варианта оформления обработчика щелчка на изображении (рис. 2.2).

1. Изображение в HTML-документе определяется, как известно, тегом <IMG>. Файл с изображением задается атрибутом SRC. Обработчик события onclick задается в примере как функция clickimageQ, которая должна быть определена где-то в контейнере <SCRIPT>. В результате щелчка на графическом изображении из файла picture.jpg выводится окно с сообщением:

```
<HTML>
<SCRIPT>
function clickimageQ {
alert("Привет!")
}
   </SCRIPT>
   <IMG SRC = "pict.jpg" onclick = "clickimageQ">
</HTML>
   Вариант 2:
<HTML>
   <IMG SRC = "pict.jpg" onclick = "alertcnpneeT!")">
</HTML>
```

Если сценарий обработки события небольшой и используется в HTML-документе один раз, то целесообразно оформлять его непосредственно в виде значения атрибута-события. В других случаях предпочтителен первый вариант, то есть оформление обработчика в виде функции.

Чтобы указать браузеру явным образом, что сценарий написан на языке JavaScript, можно в значении атрибута-события написать префикс "javascript:". Например, <IMG onclick = "javascript: aleИ:(" Привет!")">. Если не указывать язык сценария, то браузер будет подразумевать JavaScript.

2. Теперь рассмотрим еще один способ оформления обработчиков событий. Прежде всего, отметим, что почти для всех тегов HTML можно помимо прочих указать еще один атрибут — ID (идентификатор). Этот атрибут принимает любые строковые значения, которые играют роль индивидуальных имен элементов в их объектном представлении. Если атрибут ID в теге используется, то для задания обработчика события можно не использовать атрибуты-события. Вместо этого в контейнере <SCRIPT> достаточно написать определение функции обработчика события, имя которой образуется по следующему шаблону:

значение_10. событие ()

Пример

```
<HTML>
<H1 ID = "Mynеас1еr">Привет всем!</H1>
<SCRIPT>
function Myheader.onclick(){
alert("Привет!")
I
</SCRIPT>
</HTML>
```

СОВЕТ

Браузер, встречая в HTML-документе тег с определенным ID, создает в объектной модели этого документа объект с таким же именем. Для этого объекта имеется метод обработки события. Название метода совпадает с названием события, но синтаксис использования метода требует, чтобы его название было написано в нижнем регистре. В связи с этим я советую вам в любом случае писать название события в нижнем регистре.

Обратите внимание на следующие два обстоятельства, важные только для рассмотренного выше способа оформления обработчиков событий:

- Элемент HTML-документа должен быть загружен раньше, чем функция-обработчик события.
- Составное имя функции-обработчика события содержит название события в нижнем регистре.

Если требуется, чтобы сценарий обработал событие независимо от того, с каким элементом оно связано, то можно воспользоваться таким составным именем функции-обработчика:

```
function document.событие(){
    ...
}
```

Пример

```
function document.onclick(){
    ...
}
```

Приведенных выше способов обработки событий (привязки обработчиков к элементам) вполне достаточно для практических нужд разработки веб-сайтов и других приложений. Однако следует упомянуть еще об одном способе, который применим для IE версии 4.0 и старше. Он основан на использовании атрибутов FOR и EVENT в теге <SCRIPT>.

Атрибуту FOR присваивается ссылка на объект, который должен реагировать на событие, а атрибуту EVENT присваивается название события. В качестве ссылки на объект обычно используется значение идентификатора объекта, то есть значение атрибута ID.

Пример

```
<HTML>
<SCRIPT FOR = "MYBUTTON" EVENT = "onclick">
alertC'Bbm щелчок")
</SCRIPT>
<BUTTON ID = "MYBUTTON">Hax<MM 3flecb</BUTTON>
</HTML>
```

Выражения в контейнере <SCRIPT> будут выполняться только при наступлении события, указанного в EVENT, которое связано с объектом, указанным в FOR. В приведенном выше примере при щелчке на кнопке (<BUTTON . . .>) будет выведено диалоговое окно с сообщением.

Рассмотренный выше способ позволяет не использовать атрибуты-события и указания обработчиков событий в тегах элементов. Однако при этом код сценария оказывается привязанным только к одному элементу. Чтобы использовать этот же код для других элементов, придется повторить его в секциях сценария (контейнерах<SCRIPT>), отдельно для каждого элемента.

В подразделе 2.4.3 мы изучим еще один способ назначения объектам обработчиков событий — с помощью сценариев.

2.3.3. Объекты, управляемые сценариями

Мы рассмотрели, где располагаются сценарии и каким образом они могут быть привязаны к событиям. Если событие происходит, то выполняется соответствующий ему сценарий-обработчик. Кроме того, сценарий можно запустить и вне всякой связи с каким бы то ни было событием. В любом случае сценарий должен что-то с чем-то делать. Предметом деятельности сценария являются объекты окна браузера и HTML-документа, загруженного в него. Параметры элементов документа, заданные с помощью атрибутов соответствующих тегов, можно изменить. Более того, можно заменить одни теги другими и даже заменить весь загруженный HTML-документ на другой. Делается это сценариями, но не напрямую с тегами и значениями атрибутов (то есть с HTML-кодами), а с представляющими их объектами.

HTML-документ отображается в окне браузера. Окну браузера соответствует объект window, а HTML-документу, загруженному в окно, соответствует объект document. Эти объекты содержат в своем составе другие объекты. В частности, объект document входит в состав объекта window. Элементам HTML-документа соответствуют объекты, которые входят в состав объекта document. Все множество объектов имеет иерархическую структуру, называемую объектной моделью. Более подробно мы рассмотрим ее в следующей главе, а сейчас остановимся на общих вопросах, связанных с объектами.

В главе 1 мы уже встречались со встроенными объектами JavaScript и объектами, создаваемыми пользователем. Напомним, что объект представляет собой своего рода контейнер для хранения информации. Он характеризуется свойствами, методами, а также событиями, на которые может реагировать. Доступ к свойствам и методам объекта осуществляется с помощью выражений вида:

```
объект.свойство
объект.метод()
```

Объекты могут находиться в отношении вложенности (подчиненности). Объект, содержащий в себе другой объект, называют родительским. Объект, который содержится в каком-нибудь объекте, называют дочерним по отношению к нему. Таким образом, устанавливается иерархия. Чтобы указать **конкретный** объект, требуется перечислить все содержащие его объекты начиная с объекта самого верхнего иерархического уровня, подобно тому как указывается полный путь к файлу на диске:

```
объект!.объект2. . . . объектМ
```

Если объект входит в состав другого объекта (является подобъектом другого), то для доступа к его свойствам и методам используют следующий синтаксис:

```
объект!.объект2 . . . . объектN. свойство
объект!.объект2. . . . объектМ.методО
```

Нередко подобъект некоторого объекта называют его свойством (так сказать, сложным свойством). В этом случае можно говорить, что свойства объектов бывают трех типов:

- просто свойства (простые свойства);
- методы (свойства-функции);
- объекты (сложные свойства, имеющие свои свойства).

Поскольку объект document является подобъектом объекта window, ссылка на HTML-документ, загруженный в текущее (активное) окно браузера, будет выглядеть следующим образом: window. document. Объектdocumentимеетметодwrite(crpoKa), позволяющий записать в текущий HTML-документ строку, содержащую просто текст или теги HTML. Чтобы применить этот метод, следует записать: window, document.write(cipoKa).

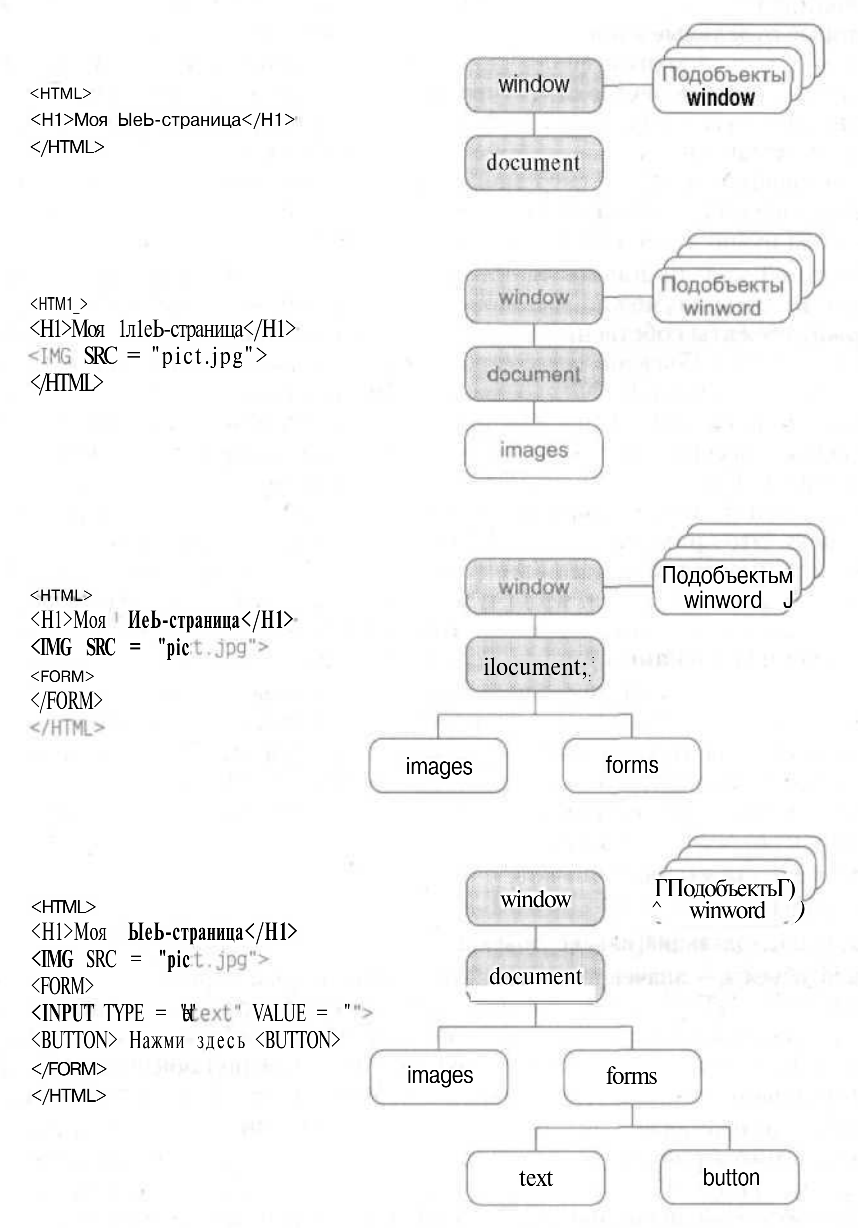


Рис. 2.3. Структуры множеств объектов для различных HTML-документов

Итак, как уже отмечалось выше, при загрузке HTML-документа в браузер создается объектная модель этого документа. Рассмотрим в общих чертах, на простых примерах, что именно происходит. Прежде всего, создается объект окна window. Это корневой объект, имеющий свои **подобъекты,** такие как location для хранения информации об URL-адресезагруженного документа и screen для хранения данных о возможностях экрана монитора пользователя. Затем создается объект document, являющийся **подобъектом** window. Далее создаются объекты, представляющие некоторые отдельные элементы HTML-документа, такие как объекты изображений, форм и их элементов (поля ввода, переключатели, кнопки) и др. На рис. 2.3 показано, как усложняется структура объектного представления документа с добавлением новых тегов. Это, конечно, упрощенные схемы, дающие лишь самые общие представления об объектной модели. Сведения об объектной модели играют роль своего рода путеводителя в обширном множестве объектов документа и окна браузера. Разработчики приложений на основе сценариев и HTML всегда имеют под рукой такой путеводитель.

В объектной модели документа объекты сгруппированы в так называемые коллекции. Коллекцию можно рассматривать как промежуточный объект, который содержит объекты собственно документа. С другой точки зрения, коллекция является массивом объектов, отсортированных в порядке упоминания соответствующих им элементов в HTML-документе. Индексация объектов в коллекции начинается с нуля. Синтаксис обращения к элементам коллекции такой же, как к элементам массива. Коллекция даже имеет свойство length — количество всех ее элементов. Так, например, коллекция всех объектов графических изображений вдокументеназываетсяimages,коллекциявсехформ — forms, коллекция всех ссылок — links. Это примеры так называемых частных или тематических коллекций. Кроме них имеется коллекция всех объектов документа, которая называется all. Один и тот же объект может входить в частную коллекцию (например, images), но обязательно входит в коллекцию all. При этом индексы этого объекта в частной коллекции и коллекции all могут быть различными.

Рассмотрим способы обращения к свойствам объектов документа. Общее правило заключается в том, что в ссылке должно быть упомянуто название коллекции. Исключением из этого правила является объект формы. Далее, если речь идет о документе, загруженном в текущее окно, то объект window можно не упоминать, а сразу начинать с ключевого слова document. Возможны несколько способов обращения к объектам документа:

- **c1оситеп^коллекциял'с1_объекта;**
- cЬситеп1.коллекция["1с1_объекта"];
- с!оситеп1.коллекция[индекс_объекта].

Здесь **id_объекта** — значение атрибута ID в теге, который определяет соответствующий элемент в **HTML-документе.** Величинаиндекс_объекта — целое число, указывающее порядковый номер объекта в коллекции. Первый объект в коллекции имеет индекс 0. Если при создании **HTML- документа** вы не использовали атрибут ID для некоторых элементов, то для обращения к их объектам остается только взять индексы. Заметим, что некоторые старые теги (например, <FORM>, <IN PUT>) имеют атрибут NAME. Значение этого атрибута можно использовать при обращении к объекту наравне со значением атрибута ID. К объекту формы, кроме описанных выше способов, можно обращаться по значению атрибута NAME (имя формы), если он указан в теге <FORM>, но не по значению атрибута ID:

```
document.имя_формы
```

Как известно, тег формы <FORM> является контейнерным и может содержать в себе теги других элементов, такие как<INPUT>, <BUTTON>И др. Обращение к элементам формы возможно через коллекцию all. Однако имеется и специфический для них способ:

```
document.имя_формы.elements[индекс_элемента]
document.имя_формы.elements[id_3neM6HTa]
document.имя_формы.  тй_элемента
```

Здесь **индекс_элемента** — порядковый номер элемента формы, а не порядковый номер в коллекции всех элементов; первый элемент имеет индекс 0. Заметим, что для Internet Explorer вместо квадратных скобок допустимо использование и круглых скобок.

Приведенный ниже HTML-код формирует простую веб-страницу, содержащую заголовок первого уровня, изображение, ссылку и форму с двумя элементами - полем ввода данных и кнопкой.

```
<HTML>
<H1>Моя веб-страница</H1>
<IMG SRC = "pict.]pg">
<A HREF = "www.admiral.ru/~dunaev">CaUT автора</a>
<FORM>
    «INPUT TYPE = "text" VALUE = "">
    <p>
    <BUTTON onclick = "ту()">Нажми здесь</BuTTOM>
</FORM>
</HTML>
```

На рис. 2.4 показано соответствие тегов элементам веб-страницы в окне браузера объектам. Поскольку ни для одного из элементов документа не указан атрибут ID (или NAME), обращение к объектам возможно посредством индексов. Так, например, для обращения к объекту графического объекта (он у нас единственный) можно использовать следующее выражение

```
document.images(0)                    // первый элемент коллекции images
```

Кнопка является вторым элементом формы. Чтобы обратиться к ней как к объекту, необходимо указать сначала форму (единственная форма в документе есть объект **forms(O))**, а затем объект этой формы **(e1 etenЛ5(индекс_элемента)).** Другими словами, достаточно записать выражение:

```
document.forms(0).elements(1)         // второй элемент формы
```

Теперь обратимся к объекту ссылки в нашем документе. Все ссылки хранятся в коллекции links. В нашем случае коллекция links содержит лишь одну ссылку. Следовательно, наша ссылка —элемент links(O) массива links. В итоге для обращения к ссылке в **HTML-**документе достаточно написать следующее выражение:

```
document.links(0)
```

Обобщая вышеизложенное, отметим, что универсальный способ обращения к объектам документа — обращение посредством коллекции all. Лично я предпочитаю именно этот способ обращения к объектам. Однако если вы не позаботились об атрибуте ID в тегах элементов **HTML-**документа, которые должны стать жертвой ваших сценариев, то обращение к объектам через all посредством индексов, особенно в случае объемистого документа, становится проблематичным. Чтобы убедиться в этом, внимательно отнеситесь к следующей задаче.

Допустим, у нас есть некоторый HTML-документ. Пусть для определенности это будет документ, рассмотренный в предыдущем примере. Интересно, какие имен-

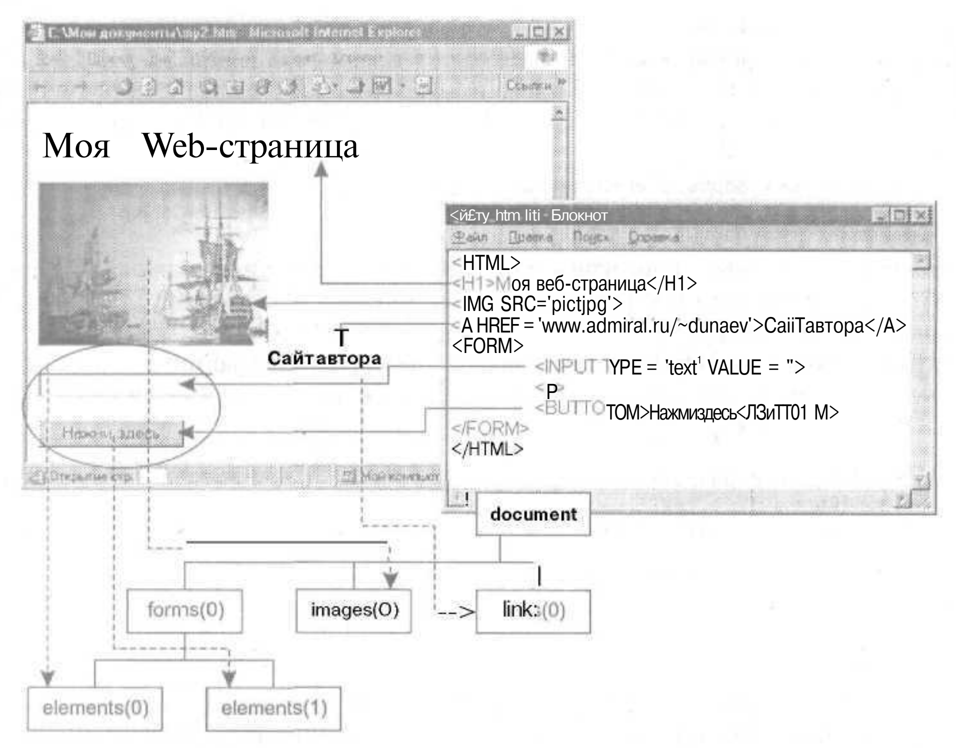


Рис. 2.4. Соответствие тегов элементам веб-страницы и объектам

но теги этого документа соответствуют элементам коллекции all объекта document? Чтобы ответить на этот вопрос, я написал простой сценарий и вставил его в конце тестируемого HTML-документа. Этот сценарий основан на использовании свойства tagName, которое возвращает название тега, с помощью которого был создан соответствующий объект. Код сценария приведен в листинге 2.1.

Листинг 2.1. Тестируемый HTML-код вместе с тестирующим сценарием

```
<HTML>
<H1>Моя веб-страница</H1>
<p>
<IMG SRC = "pict.jpg">
<p>
<A HREF = "www.admiral.ru/~dunaev">Сайт автора</a>
<p>
<FORM>
    <INPUT TYPE = "text" VALUE = "">
    <p>
    <BUTTON onclick = "ту()">Нажми здесь</BUTTON>
</FORM>
<SCRIPT>
msg = ""
for(i = 0; i < document.all.length; i++){
msg += i + " " + document.all[i].tagName + "\n"
}
alert(msg)
```

```
</SCRIPT>
</HTML>
```

В результате выполнения этого HTML-кода окно браузера будет выглядеть, как показано на рис. 2.5. В диалоговом окне, выведенном с помощью alertQ, перечислены теги HTML-документа. Их порядковые номера соответствуют индексам в коллекции all. Например, объекту document.all(5) соответствует элемент HTML-документа, созданный тегом <IMG>. Обратите внимание, что хотя теги <HEAD>, <TITLE> и <BODY> в нашем документе явно не упоминаются, соответствующие им объекты присутствуют в объектной модели этого документа и, следовательно, в коллекции all

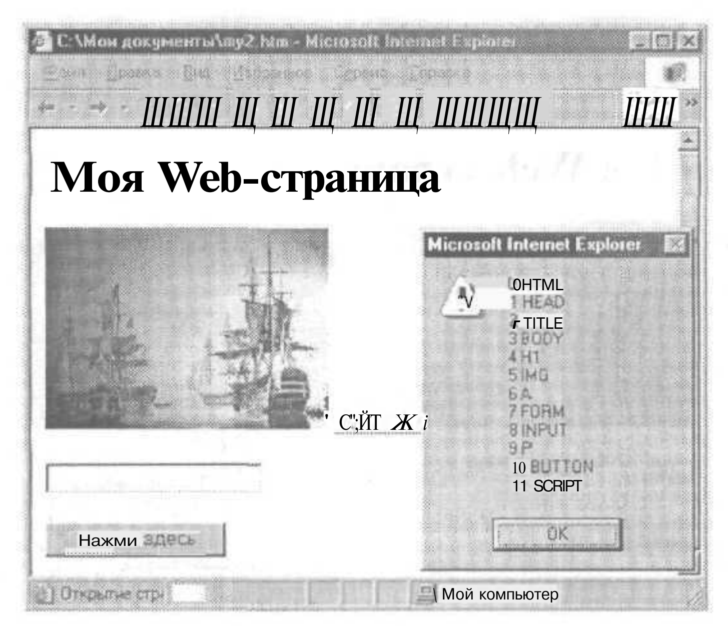


Рис. *2.5.* В диалоговом окне перечислены теги HTML-документа и их индексы в коллекции all

Теперь рассмотрим тот же пример, но добавим к основным тегам атрибуты ID, с помощью которых можно персонифицировать соответствующие объекты. Как и раньше, мы добавим к HTML-документу сценарий, выводящий окно с информацией о тегах. При этом, кроме порядкового номера и названия тега, мы будем выводить и значение атрибута ID (рис. 2.6). Чтобы получить значение атрибута ID, необходимо воспользоваться свойством id соответствующего объекта. HTML-код приведен в листинге 2.2.

Листинг 2.2. Тестируемый код с добавлением атрибутов ID

```
<HTML>
<H1>Моя веб-страница</H1>
<IMG ID = "myimage" SRC = "pict.jpg">
<A ID = "myref" HREF = "www.admiral.ru/~dunaev">CaUT автора</a>
<FORM ID = "myform">
   <INPUT ID = "myinput" TYPE = "text" VALUE ="qewe">
   <p>
   <BUTTON ID = "mybutton" onclick = "ту()">Нажми здесь</B1ЯТСШ>
</FORM>
```

продолжение &

Листинг 2.2 *(продолжение)*

```
<SCRIPT>
msg = ""
for(i =0; i<document.all.length; i ++){
 msg+= i + " " + document.all[i].tagName + " id = "+ document.all[1].id
 + "\n"
}
alert(msg)
</SCRIPT>
</HTML>
```



Рис. 2.6. Теги HTML-документа, их индексы в коллекции **all,** а также значения атрибутов ID

Итак, универсальный способ обращения к объекту документа базируется на использовании коллекции all. Однако кроме него в IE5+ и NN6 существует еще один способ, основанный на методе getElementByldQ:

```
document.getElementBy Id(значение_Ю)
```

Обратите внимание, в каком регистре пишутся отдельные буквы этого метода. Метод getElementById(значение_Ю) позволяет обратиться к любому элементу по значению его идентификатора — значению атрибута ID. Если несколько элементов документа имеют одинаковый ID, метод возвращает первый элемент с указанным значением ID. Отсюда видно, насколько важно присваивать различным элементам уникальные значения ID.

Выше мы рассмотрели различные способы обращения к объектам. Ссылки на объекты часто используются, главным образом, как промежуточный этап для доступа к конечным свойствам и методам. В конце концов, именно они и интересуют нас при создании сценариев. Заметим попутно, что ссылки на объекты можно сохранять в переменных, которые потом можно использовать для обращения к свойствам и методам.

В приведенном выше примере элемент изображения, созданный с помощью тега <IMG ID = "myimage" SRC = "pict.jpg">, имеет атрибуты ID и SRC. Этим атрибутам соответствуют свойства id и src объекта этого изображения. Ниже приведены примеры различных вариантов обращения к данным свойствам:

```
document.images(0).id          // "myimage"
document. images ("myimage") . id
document.all("myimage").id
document.all.myimage.id
var x = document.all.myimage   // объект
x.id                            // "myimage"
document.images(Q).src        /* строка URL, например,
                                  f П e:///C:/Мои%20документы/pict.jpg */
document.images("my image").src
document.al l("myimage").src
document.all.myimage.src
var x = document.all.myimage   // объект
x.src                          /* строка URL, например,
                                  file:///C:/Мои%20документы/p1ct . jpg */
```

Объекты документа имеют много интересных и полезных свойств. Одни из них доступны только для чтения и могут использоваться в сценариях в качестве информации для выполнения каких-то действий. Другие свойства можно изменять, то есть присваивать им иные значения (например, для изменения рисунка достаточно изменить значение свойства src). В следующей главе мы рассмотрим, для чего служат конкретные свойства и методы.

2.4. Понятие события

Каждое действие пользователя (нажатие на клавишу, щелчок кнопкой мыши и т. п.) формирует некоторое событие, то есть сообщение о произошедшем. Операционная система (например, Windows) анализирует это сообщение, чтобы узнать, откуда оно взялось и что с ним делать дальше. Если, например, пользователь нажал на кнопку мыши в момент, когда ее указатель находился над окном браузера, то Windows пошлет браузеру сообщение о том, какая кнопка мыши была нажата, какие при этом клавиши клавиатуры удерживаются, а также координаты указателя мыши. Если пользователь щелкнул где-то на панели инструментов, браузер отработает это сообщение сам. Если же в момент щелчка указатель находился на «территории» HTML-документа, то браузер пропустит сообщение о событии через свою объектную модель. В HTML-коде документа может находиться сценарий для обработки этого события. Инструкции этого сценария направляются к браузеру опять же через объектную модель.

2.4.1. Свойства события

Сообщение о событии формируется в виде объекта, то есть контейнера для хранения информации. Как только объект события создан, браузер присваивает значения его свойствам. Например, объект, соответствующий щелчку мышью, содержит координаты указателя мыши, а также сведения о том, какая кнопка была нажата. Кроме того, объект события в одном из своих свойств содержит ссылку на элемент, с которым связано данное событие (например, на кнопку, изображение, поле ввода и т. п.).

Объект события хранится в памяти столько времени, сколько необходимо сценарию для его обработки. Пока выполняется обработчик события, объект события вместе со своими свойствами доступен сценарию, находящемуся в памяти браузера. Как только обработчик события завершит работу, объект события становится пустым (возвращается в исходное состояние).

В любой момент времени существует не более одного объекта события. Все инициированные события заносятся операционной системой в буфер и выполняются последовательно в том порядке, в каком они туда попали.

В объектной модели имеется объект event, являющийся подобъектом объекта окна window. Он содержит информацию о том, какое событие произошло, какой элемент должен на него реагировать, и ряд других характеристик. Объект event можно использовать в сценариях для документов с большим количеством интерактивных элементов, для выяснения причин неправильного отображения веб-страниц. Далее мы рассмотрим некоторые наиболее часто используемые свойства объекта event.

В качестве примера ниже приводится HTML-документ, не содержащий видимых элементов. Щелчок мышью или нажатие клавиши на клавиатуре выводит диалоговое окно со значениями некоторых свойств объекта event (рис. 2.7). Так, свойства x и y содержат значения координат указателя мыши в момент щелчка, отсчитанные в данном случае относительно окна браузера. Свойство keyCode возвращает код нажатой клавиши с символам в системе Unicode. Для латинских и цифровых символов кодировки Unicode и ASCII совпадают. При нажатии навигационных клавиш и клавиш дополнительной цифровой клавиатуры keyCode возвращает null. По существу, keyCode — это код соответствующего символа, а не код клавиши. Для фиксации самой клавиши используются события onkeydown и onkeyup, а не on keypress.

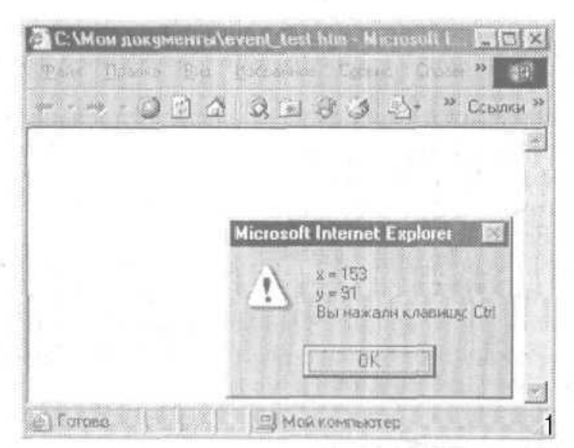


Рис. 2.7. Щелчок в окне браузера выводит сообщение о координатах указателя мыши и нажатой клавише

```
<HTML>
<BODY ID = "test">
</BODY>
<SCRIPT>
function test .onclickC) {
var str=""
str += "x = " + window.event.x + "\n"
```

```
str += "y = " + window.event.y + "\n"
str += "Вы нажали клавишу: "
if (window.event.shiftKey){str += "Shift"}
if (window.event.ctrlKey){str *= "Ctrl"}
if (window.event.altKey){str += "Alt"}
alert(str)
)

function test .onkeypressO{
alert("Код клавиши: " + window.event.keyCode)
I
</SCRIPT>
</HTML>
```

В IE версии 4.0 и старше среди прочих часто оказываются полезными свойства button и srcElement.

Свойство button возвращает целочисленное значение, указывающее, какая кнопка или кнопки мыши были нажаты (табл. 2.1).

Таблица 2.1. Значения свойства button

Значение	Описание
0	Кнопки не нажаты
j	Нажата левая
?	Нажата правая
3	Одновременно нажаты левая и правая
4	Нажата средняя
5	Нажаты левая и средняя
6	Нажаты правая и средняя
7	Все три кнопки нажаты

Свойство srcElement возвращает ссылку на объект элемента HTML-документа, который инициировал событие. При получении такой ссылки можно узнать или изменить значения свойств этого объекта и применить к нему любой из его методов.

Приведенный ниже HTML-код формирует документ, содержащий две кнопки (рис. 2.8). Сценарий обрабатывает событие **onclick** — щелчок мышью. Он привязан к элементу, заданному тегом <BODY>. Щелкнуть можно на любой кнопке, а также на незанятом месте окна браузера (рис. 2.9). В любом случае событие onclick будет обработано сценарием (функцией **changetextQ),** поскольку кнопки заданы внутри тега **<BODY>.** Сценарий заменит текст, который находится внутри тега элемента, инициировавшего событие. Какой именно элемент инициировал событие, определяется с помощью свойства srcElement, а замена текста осуществляется посредством свойства innerText. Об этом свойстве мы еще поговорим отдельно. Обратите внимание, что результат зависит от того, где находился указатель мыши в момент щелчка.

```
<HTML>
<BODY onclick = "changetext()">
<button>nepsa« KHonKa</button>
<Ьи11оп>Вторая KHonKa</button>
</BODY>
```

```
<SCRIPT>
function changetext(){
x = window.event.srcElement            // ссылка на объект
x.innerText = "Уже щелкнули"           // замена текста
}
</SCRIPT>
</HTML>
```

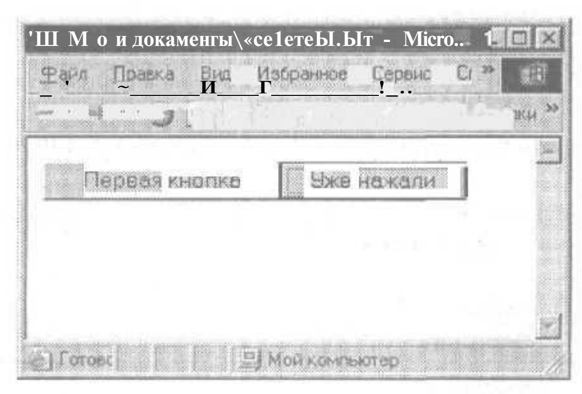


Рис. 2.8. Результат щелчка на второй кнопке

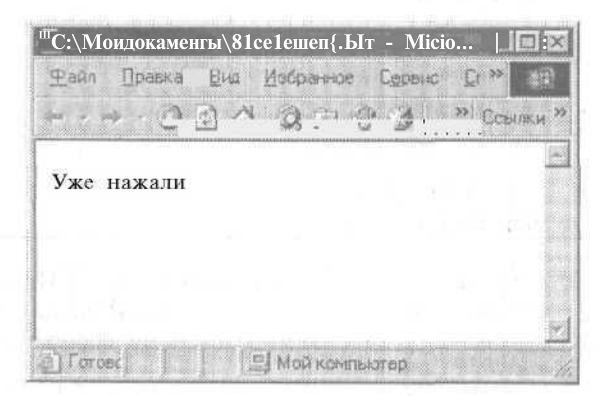


Рис. 2.9. Результат щелчка на свободном месте

Чтобы на щелчок реагировали только кнопки, приведенный выше код необходимо несколько модифицировать:

```
<HTML>
<BODY>
<button onclick = "changetext()">Первая кнопка</button>
<button onclick = "changetext()">Вторая кнопка</button>
</BODY>
<SCRIPT>
function changetext(){
x = window.event.srcElement            // ссылка на объект
x.innerText = "Уже щелкнули"           // замена текста
}
</SCRIPT>
</HTML>
```

Объект event имеет несколько свойств, содержащих координаты в пикселах указателя мыши в момент возникновения события, связанного с мышью. Многообразие типов этих координат обусловлено различиями в начале их отсчета. Рассмотрим эти свойства по порядку.

- screenX, screenY — координаты указателя мыши относительно левого верхнего угла экрана. Например, если экран монитора находится в режиме 800x600, то координаты правого нижнего угла экрана будут равны 800 и 600 пикселов соответственно.
- clientX, clientY — координаты указателя мыши относительно области клиента (рабочей области браузера), в которой находится HTML-документ, без учета рамок, полос прокрутки, меню и т. п.
- offsetX, offsetY — координаты указателя мыши относительно верхнего левого угла элемента, инициировавшего событие.
- x, y — координаты указателя мыши относительно верхнего левого угла первого абсолютно или относительно позиционированного контейнера, в котором находится элемент, инициировавший событие. Позиционированный контейнер — это элемент, заданный каким-нибудь контейнерным тегом (например, <BODY>, <DIV>, <H1>) с атрибутом STYLE, для которого указано свойство position. Если такого контейнера нет, то свойства x и y возвращают координаты относительно главного документа, такие же, как и clientX, clientY.

Чтобы лучше разобраться в этих свойствах, создайте тестовый HTML-документ, как показано ниже, и поэкспериментируйте с ним, щелкая мышью в различных точках (рис. 2.10). Этот документ содержит два контейнера, созданных с помощью позиционированных различным образом тегов <DIV>. Первый контейнер позиционирован абсолютно, а второй, вложенный в него, — относительно. Сценарий определяет значения координат различных типов и отображает их в диалоговом окне.

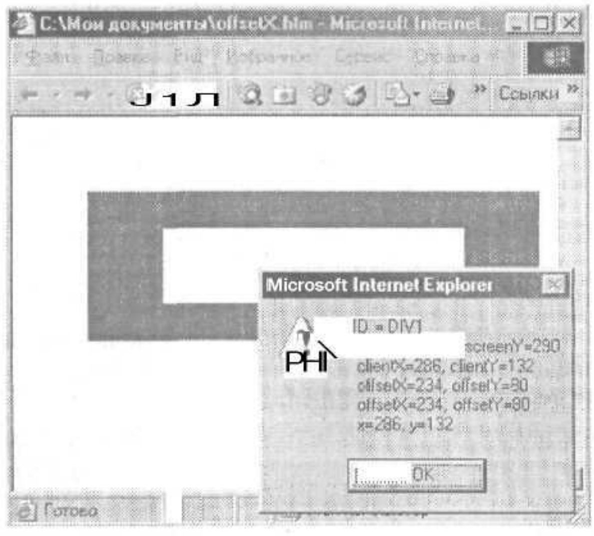


Рис. 2.10. При щелчке выводится окно с различными типами координат указателя мыши

```
<HTML>
<BODY ID = "Mybody"
<DIV ID = "DIV1" STYLE = "position:absolute; left:50; top:50; width:300;
  height:100; background-color:blue">
<DIV ID = "DIV1" STYLE = "position:relative; left:50; top:25; width:200;
  height:50; background-color:yellow">
</DIV>
</DIV>
<BODY>
<SCRIPT>
function document.onclick(){
var e = window.event                         //ссылка на объект event
var str = "ID = " + e.srcElement.id +
 "\n screenX= " + e.screenX + ", screenY= " + e.screenY +
"\n clientX=" + e.clientX + ", clientY=" + e.clientY +
"\n offsetX=" + e.offsetX + ", offsetY=" + e.offsetY +
"\n offsetX=" + e.offsetX + ", offsetY=" + e.offsetY +
"\n x=" + e.x + ", y=" + e.y
alert(str)
}
</SCRIPT>
</HTML>
```

Нам требовалось, чтобы на щелчок реагировал любой объект документа, поэтому мы не стали привязывать обработчик события индивидуально ко всем элементам, а вместо этого назвали функцию-обработчик составным именем document.onclick().

Обратите внимание на использование свойства srcElement для получения значения id элемента, инициировавшего (или, как еще говорят, получившего) событие.

У объекта event имеется изменяемое свойство returnValue (возвращаемое значение). С его помощью можно отменять действия, принятые по умолчанию. Для этого достаточно присвоить ему значение false. Например, щелчок на ссылке по умолчанию означает переход по указанному адресу. Однако вы можете отменить это действие или запросить у пользователя подтверждение перехода.

Пример

```
<HTML>
<A ID = "myref" HREF = "www.chat.ru">Переход на chat.ru</A>
<SCRIPT>
  function myref.onclick(){
  ret = confirm("Вы действительно хотите перейти?")
  if (!ret)
  window.event.returnValue = false
  }
</SCRIPT>
</HTML>
```

Здесь метод confirm() выводит диалоговое окно. Если пользователь не подтвердил переход, то **confirm()** возвращает false, а свойству returnValue тоже присваивается false. В результате переход по ссылке не произойдет.

Нередко требуется разместить в документе элемент, который внешне выглядит, как обычная ссылка (при наведении указателя мыши его вид изменяется), но щелчок должен приводить не к переходу на другой документ, а просто к выполнению некоторого сценария. Этого можно добиться несколькими способами. Один из них основан на использовании свойства returnValue:

```
<HTML>
<A ID = "myref" HREF = "">Щелкни здесь</A>
<SCRIPT>
   function  myref.onclick(){
   alert("5bm  щелчок  на  ссылке")
   window.event.returnValue  =  false
   }
</SCRIPT>
</HTML>
```

Если бы мы не использовали оператор window.event.returnValue = false, то из-за неуказанного в HREF адреса перехода по ссылке после выполнения обработчика события открылось бы новое окно браузера с содержимым текущей папки.

Другой способ отключить действие щелчка на ссылке, принятое по умолчанию, заключается в том, чтобы в значении атрибута HREF написать "T', как в следующем примере:

```
<A ID = "myref" HREF = "#" onclick="myclick()">Щелкни здесь</A>
```

2.4.2. Прохождение событий

Как уже отмечалось выше, для элементов документа можно указать события, на которые они должны реагировать, и обработчики этих событий. Например, если требуется, чтобы элемент реагировал на щелчок кнопкой мыши выполнением некоторой функции myfunc(),то в тег этого элемента следует вставить запись: onclick = "myfuncO". Однако нам известно, что большинство тегов в HTML являются контейнерными и, следовательно, могут содержать в себе другие теги. При этом может оказаться так, что одно и то же событие будет обозначено в различных, но вложенных друг в друга тегах. Что произойдет при наступлении этого события? Как оно распространяется по объектам и как оно перехватывается элементами документа?

Рассмотрим в качестве примера следующий HTML-документ, содержащий единственную кнопку:

```
<HTM1_>
<BODY onclick = "alert('Щелчок на body')">
<BUTTON onclick = "alert('Щелчок на button') ">Нажми здесь</ВиТТОМ>
</BODY>
</HTML>
```

Особенность этого документа в том, что одно и то же событие (с различными обработчиками) привязано к различным тегам, один из которых содержит в себе другой (<BUTTON> содержится в <BODY>). Щелчок на кнопке приведет к выполнению сначала обработчика щелчка для кнопки, а затем обработчика для <BODY>. Если щелкнуть где-нибудь в рабочей области окна браузера вне кнопки, то сработает только обработчик для <BODY>. Ниже мы рассмотрим, как можно избавиться от этого нежелательного эффекта.

Модели прохождения событий в Internet Explorer и Netscape Navigator отличаются друг от друга. Более того, имеются различия между этими моделями для различных версий одного и того же браузера даже начиная с IE4 и NN4.

Модель прохождения событий в IE4 называется «всплыванием событий». События, подобно воздушным пузырькам в воде, как бы всплывают от целевого объек-

та самого нижнего уровня вверх по иерархии объектов. В рассмотренном выше примере событие onclick «всплывает» от объекта кнопки к объекту тела документа. В NN4 модель прохождения событий называют захватом событий. Согласно этой модели событие распространяется, наоборот, от самого верхнего в иерархии объекта window к целевому объекту. Чтобы события обрабатывались на уровне текущих объектов, требуется включить для них режим захвата. Это делается с помощью специального метода captureEventQ. В IE4 для управления всплыванием событий используют свойство прерывания всплывания cancelBubble.

В современных браузерах IE5.5+ и NN6 реализована модель, объединяющая в себе и всплывание, и захват событий. Всплывание организовано как в IE4+, а захват — как в NN4. По умолчанию событие всплывает, однако вы можете включить его захват. Тогда событие сначала достигает целевого объекта, а затем начинает всплывать в обратном направлении.

В IE4+ управлять прохождением событий можно с помощью свойства cancelBubble объекта события event. Когда в документе (на веб-странице) происходит какое-либо событие, объект event первым получает информацию о нем и решает, какому элементу его передать. Например, когда объект event получает событие onclick (щелчок кнопкой мыши), он выясняет, какой элемент находился в тот момент под указателем мыши. Если там было два элемента (один над другим), то выбирается элемент с наименьшим z-индексом, то есть нижний. Напомню, что z-index является одним из параметров позиционирования элементов с помощью таблиц стилей. Установив элемент, связанный с событием, объект event ищет обработчик этого события и выполняет его. Например, если элемент имеет уникальное имя (ID) Myelement, то ищется функция Myelement.onclickQ. Далее объект event выясняет, какой объект является контейнером для данного элемента. Если таковой имеется, то событие переходит внутрь этого контейнера. Если, например, контейнером является объект, созданный тегом <DIV> с именем Mydiv, то будет предпринята попытка выполнить функцию Mydiv.ondickQ. Этот процесс продолжается, пока имеются доступные контейнеры. Например, последним доступным контейнером может оказаться документ (объект document), и тогда объект event ищет функцию document.onclickQ.

Чтобы определить главного виновника события (объект, инициировавший событие), используется свойство srcElement. Мы уже говорили о нем выше. Например, может потребоваться одинаковая реакция на одно и то же событие всех элементов, за исключением некоторых. В таких случаях для элементов, на которых процесс всплывания события должен закончиться, создаются специальные функции-обработчики. В этих функциях помимо прочего должно быть выражение присвоения свойству cancelBubble значения true, чтобы прервать дальнейшее прохождение события. Например, если мы хотим разорвать цепную передачу события на элементе с именем Myelement, необходимо создать для него функцию-обработчик события следующего вида:

```
function Myelement. onclickQ{
        // код обработки события onclick
   window.event. cancelBubble = true
   }
```

Делать такое в функции-обработчике для объекта document бессмысленно, поскольку выше событие все равно всплыть не может, а отменить реакцию на это событие элемента-инициатора (первого звена цепочки) невозможно.

В начале этого подраздела мы уже рассматривали пример HTML-документа с кнопкой, вложенной в контейнер <BODY>:

```
<HTML>
<BODY onclick = "alert('Щелчок на body')">
<BUTTON onclick = "alert ('Щелчок на button')">Нажми здесь</BUTTON>
</BODY>
</HTML>
```

Особенность этого примера в том, что если пользователь щелкнет на кнопке, сработает обработчик не только кнопки, но и тела документа (элемента, заданного тегом <BODY>. Если мы хотим это предотвратить, то можно переписать код следующим образом:

```
<HTML>
<BODY onclick = "alert('Щелчок на body')">
<BUTTON onclick = "alert('Щелчок на button');
window.event .cancelBubble = true ">Нажми здесь</B11TTOM>
</BODY>
</HTML>
```

Возможен также и такой вариант кода:

```
<HTML>
<BODY onclick = "alert('Щелчок на body')">
<BUTTON ID = "MyБи'">Нажми здесь</B1ЛTOM>
</BODY>
<SCRIPT>
function Mybut.onclick() {
alert('Щелчок на button')
window.event.cancelBubble = true      // прекратить всплывание события
I
</SCRIPT>
</HTML>
```

2.4.3. Указание обработчика события в сценарии

Об обработчиках событий достаточно подробно рассказывалось в предыдущем разделе. Там так или иначе обработчики событий привязывались к специальным атрибутам HTML-тегов — ID или атрибутам-событиям — FOR и EVENT. Сценарий-обработчик выполнялся, если возникало событие, связанное с элементом, для которого был указан этот обработчик. Вместе с тем имеется возможность вызывать обработчик & сценарии просто как метод объекта. Он будет выполняться так же, как и при возникновении соответствующего события.

Пример

```
<HTML>
<BUTTON ID="Mybutton" onclick="butclick()">Нажми здесь</ВиTTOM>
<SCRIPT>
document.all.Mybutton.onclick()          // вызов обработчика
function butclick(){
alert('Щелчок на кнопке Mybutton')
I
</SCRIPT>
</HTML>
```

В данном случае сразу же после загрузки документа в браузер вызывается обработчик события onclick, связанного с кнопкой. Этот обработчик — функция butclick(). Обратите внимание, что обработчик выполнится несмотря на то, что

самого щелчка не было. Заметим, что выражение вызова метода (в данном случае это o.onctickQ) должно записываться только строчными буквами.

Следует иметь в виду, что рассмотренный выше способ использования обработчика как метода существенно отличается от эмуляции действия для события. Например, допустим, что имеется форма с несколькими полями для ввода данных, одно из которых имеет обработчик события onfocus (активизация поля). Чтобы поле активизировалось, необходимо щелкнуть на нем мышью или перейти к нему с помощью клавиатуры. При этом сработает обработчик события onfocus. Однако даже если поле не активно, обработчик события onfocus все равно можно вызвать, но при этом поле не активизируется:

```
document.имя_формы.имя_поля.onfocusO
```

Благодаря возможности использовать обработчик события в качестве метода объекта можно задавать обработчик в виде функции, имеющей название, совпадающее с названием события:

```
<HTML>
<BODY onload = "onloadQ"
...
</BODY>
<SCRIPT>
function onload() {
...
>
</SCRIPT>
</HTML>
```

2.5. Работа с окнами и фреймами

Как известно, HTML-документ загружается в окно браузера. Можно открыть несколько таких окон и загрузить в них различные документы, а также разбить одно окно на несколько прямоугольных областей, называемых фреймами. В каждый такой фрейм можно загрузить отдельный документ. При этом существует возможность организовать взаимодействие между фреймами — например, с помощью действий в одном фрейме управлять содержимым другого фрейма.

Загрузка в браузер HTML-документа приводит к тому, что в браузере создается иерархическая объектная модель этого документа, на самом верхнем уровне которой находится объект window. Доступ к свойствам и методам данного объекта имеет уже знакомый вам синтаксис:

```
window.свойство
wi ndow.методе[параметры])
```

У объекта window имеется синоним self, используемый при обращении к окну, содержащему текущий документ. Иначе говоря, идентификатор self применяется в многооконных или многофреймовых системах, когда требуется указать окно с документом, в котором находится данный сценарий. Его рекомендуется вставлять, чтобы не запутаться. При запуске сценария в ссылках на объекты только текущего документа (типичная ситуация) идентификаторы window и self можно опускать.

У объекта window есть ряд подобъектов. Мы уже рассматривали некоторые свойства объекта события event, который является подобъектом объекта window. Другой объект, location, содержит информацию, полезную для работы в сети и для создания ссылок в документах с многофреймовой структурой. Кроме этого, свойство href объекта location используется для загрузки документа в текущее окно:

```
window, location. href = "иК1_-адрес_документа"
```

Данный способ загрузки документов в текущее окно браузера доступен во всех версиях IE и NN. В IE можно использовать также и метод navigateQ:

```
window.navi gate("URL-aflpec_flOKyMeHTa")
```

2.5.1. Создание новых окон

Главное окно браузера создается не с помощью сценариев, а автоматически, когда пользователь запускает браузер, а также при открытии документа с определенным URL-адресом или другого файла. В HTML открыть документ в новом окне можно с помощью атрибута TARGET тега ссылки <A HREF = ... >. Например, <A HREF = "http://www.rambler.ru" TARGET = "newWindow"> Rumbler</A>.

С помощью сценария можно создать любое количество окон. Для этого применяется метод open():

```
wi ndow.open([параметры])
```

Этому методу передаются следующие необязательные параметры:

- адрес документа, который нужно загрузить в создаваемое окно;
- имя окна (как имя переменной);
- строка описания свойств окна (features).

В строке свойств записываются пары свойство=значение, которые отделяются друг от друга запятыми. В табл. 2.2 приведен список свойств окна, передаваемых в строке features. Значения yes и по можно заменить числовыми эквивалентами 1 и 0 соответственно.

Таблица 2.2. Свойства окна, передаваемые в строке features

Свойство	Значения	Описание
channel mode	yes, no, l, 0	Показывает элементы управления Channel
directories	yes, no, 1, 0	Включает кнопки каталога
fullscreen	yes, no, 1, 0	Полностью разворачивает окно
height	Число	Высота окна в пикселах
left	Число	Положение по горизонтали относительно левого края экрана в пикселах
location	yes, no, 1, 0	Текстовое поле Address
menubar	yes, no, 1, 0	Стандартные меню браузера
resizeable	yes, no, 1, 0	Может ли пользователь изменять размер окна
scrollbars	yes, no, 1, 0	Горизонтальная и вертикальная полосы прокрутки
status	yes, no, 1, 0	Стандартная строка состояния

продолжение &

Таблица 2.2 *(продолжение)*

Свойство	Значения	Описание
toolbar	yes, no, \, 0	Включает панели инструментов браузера
top	Число	Положение по вертикали относительно верхнего края экрана в пикселах
width	Число	Ширина окна в пикселах

Примеры

```
window.open("mypage.htm","NewWin", "height=150, width=300")
window.open("mypage.htm")

strfeatures = "top=100,left=15,width=400, height=200, location=no,
  menubar=no"
window.open("www.admiral.ru/~dunaev", "Сам себе веб-дизайнер",
strfeatures)
```

Вместо третьего параметра (строки features) можно использовать значение true. В этом случае указанный документ загружается в уже существующее окно, вытесняя предыдущий. Например, **window.open("mypage.htm","NewWin",** true).

Метод **window.open** () возвращает ссылку на объект окна. Эту ссылку можно сохранить в переменной, чтобы потом использовать, например при закрытии окна.

Для закрытия окна служит метод **close()**. Однако выражения **window.close()** или self.close() закрывают главное окно, а не дополнительное, которое вы создали методом open(). В этом случае как раз и необходима ссылка на созданное окно. Эту ссылку следует сохранить в глобальной переменной, чтобы иметь доступ к ней до тех пор, пока главный документ загружен в браузер. Вот пример:

```
var objwin = window.open("mypage.htm", "Моя страница")
objwin.close()
```

Метод Window.open() открывает новое независимое окно как экземпляр браузера. В этом случае при закрытии главного окна браузера новое окно остается открытым. Независимые окна называют еще немодальными (modalless). Однако можно создать и так называемое модальное окно. Пока открыто модальное окно, пользователь не может обратиться к другим окнам, в том числе и к главному. Так обычно работают стандартные диалоговые окна. Например, окна, создаваемые методами alert(), prompt() и **confirm()**, являются модальными. В модальное окно можно загрузить любой документ.

Для создания модального окна используется метод **showModalDialog()**. Так же, как и метод **open()**, он принимает в качестве параметров адрес документа (файла), имя окна и строку свойств. Однако формат этой строки другой. В частности, параметры в строке разделяются точкой с запятой, размеры окна и координаты его верхнего левого угла требуют указания единиц измерения (например, px — пикселы). Кроме того, этот метод не возвращает ссылку на объект окна, поскольку она не нужна для модального окна.

В табл. 2.3 приведен список свойств окна, созданного методом **showModalDialog()**, передаваемых в строке features.

Таблица 2.3. Свойства окна, созданного методом showModalDialogQ

Свойство	Значения	Описание
order	thick, thin	Размер рамки вокруг окна (толстая/тонкая)
center	yes, no (1, 0)	Выравнивание окна по центру главного
dialogHeight	Число + единицы измерения	Высота окна
dilalogLeft	Число + единицы измерения	Горизонтальная координата
dialogTop	Число + единицы измерения	Вертикальная координата
dialogWidth	Число + единицы измерения	Ширина окна
font	Строка таблицы стилей	Стиль, определенный по умолчанию для окна
font-family	Строка таблицы стилей	Вид шрифта, определенный по умолчанию для окна
font-size	Строка таблицы стилей	Размер шрифта, определенный по умолчанию для окна
font-style	Строка таблицы стилей	Тип шрифта, определенный по умолчанию для окна
font-variant	Строка таблицы стилей	Вариант шрифта (обычный/курсив), определенный по умолчанию для окна
font-weight	Строка таблицы стилей	Толщина шрифта, определенная по умолчанию для окна
help	yes, no, 1, O	Включение кнопки Help в верхнюю панель
maximize	yes, no, 1, O	Включение кнопки Maximize в верхнюю панель
minimize	yes, no, 1,0	Включение кнопки Minimize в верхнюю панель

Пример

```
<HTML>
<BUTTON onclick = "return OpenWinlO ">0ткрыть окно 1 </BUTTON>
<BUTTON onclick = "return OpenWin2 () ">0ткрыть окно 2 </BUTTON>
<SCRIPT>
var newWindow                    // глобальная переменная для ссылки на окно

function OpenWinl(){             // открытие 1-го окна
window. status = "Первое окно"   // статусная строка главного окна
strfeatures= "top=100, lett=50,width=300,height=270,toolbar=no"
window.open ("1.htm","winl",strfeatures)
I
function OpenWin2(){             // открытие 2-го модального окна
window. status = "Второе окно    // статусная строка главного окна
strfeatures = "dialogWidth=500px;dialogHeight=320px , border=thin ; help=no"
window.showModalDialog("2.htm","win2", strfeatures)
}
function CloseWinl() {           // закрытие 1-го окна
if (newWindow){                  // если 1-е окно открыто
newWindow .close()               // закрыть 1-е окно
```

```
newWindow = null                    // очистить ссылку на 1-е окно
window.status = ""
}
</SCRIPT>
</HTML>
```

2.5.2. Фреймы

Фрейм представляет собой прямоугольную область окна браузера, в которую можно загрузить HTML-документ. Разбиение окна на фреймы происходитс помощью HTML-кода, содержащегося в отдельном HTML-файле, который называется установочным. Установочный файл содержит только теги разметки фреймов, то есть их относительное расположение, размеры, ссылки на загружаемые в них документы и другие параметры фреймов. В нем не должны присутствовать теги других элементов и текстовая информация, за исключением разве что тегов <META>. В установочном файле можно написать сценарий, создающий фреймовую структуру документа.

Напомним, что разбиение окна браузера на несколько фреймов обычно производится с помощью тега <FRAMESET>. Внутрь этого тега вставляются теги <FRAME> с атрибутами, указывающими имя фрейма и адрес HTML-документа, который будет отображаться в этом фрейме. В следующем примере создается два фрейма, расположенные друг над другом:

```
<HTML>
<FRAMESET ROWS="30%,70%"
<FRAME 5ИС="документ1.Шт" NAME="framel">
<FRAME 5КС="документ2.Шт" NAME="framel">
</FRAMESET>
</HTML>
```

Это так называемое вертикальное расположение фреймов. Если вместо атрибута ROWS в теге <FRAMESET>использовать атрибут COLS, то фреймы будут расположены горизонтально: второй фрейм находится справа от первого. Используя вложение тега <FRAMESET>, можно разбить уже имеющийся фрейм на другие два фрейма и т. д.

Отношения между фреймами и главным окном

При разбиении окна на два фрейма в объектной модели браузера возникает иерархия объектов, в которой главное окно браузера представляется в виде главного, родительского фрейма, а созданные два фрейма являются дочерними фреймами, или, иначе, фреймами-потомками. При разбиении любого из этих фреймов на следующие два фрейма последние являются дочерними для исходного. Итак, между фреймами возникает отношение родители-потомки (родители-дочки).

При запуске браузера объект главного окна window формируется автоматически. Если в это окно загружается документ с фреймовой структурой, то главное окно получает статус родительского фрейма по отношению ко всем остальным. С другой стороны, каждый тег<FRAME> внутри контейнера <FRAMESET> создает свой собственный объектwindow,вкоторый загружается соответствующийдокумент. Каждому из фреймов соответствует свой объект document. С точки зрения объекта

document ему соответствует единственный контейнер — фрейм. Хотя родительский объект и не виден пользователю, он все равно присутствует в объектной модели.

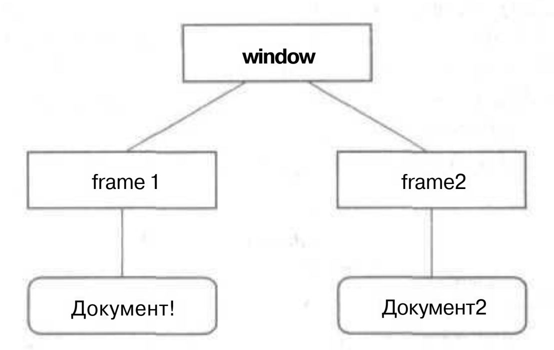


Рис. 2.11. Родительское окно и два фрейма

Итак, в вершине иерархии находится окно браузера window. Оно поделено на два фрейма с именами, например framel и frame2. Объект window является родительским по отношению к дочерним framel и frame2 (рис. 2.1 1). Для ссылки на родительское окно используется ключевое слово parent.

Допустим, пользователь активизирует некоторую ссылку в первом фрейме, но соответствующий документ должен загрузиться не в этот же фрейм, а в другой. Для решения этой задачи необходимо рассмотреть три случая:

- главное окно получает доступ к фрейму-потомку;
- фрейм-потомок получает доступ к родительскому (главному) окну;
- фрейм-потомок получает доступ к другому фрейму-потомку.

Между объектом window и объектами framel и frame2 существует прямая связь родитель-потомок. Поэтому если вы пишете сценарий в установочном файле, создающем эти фреймы, то к фреймам можно обращаться по имени, как это показано на рис. 2.12.

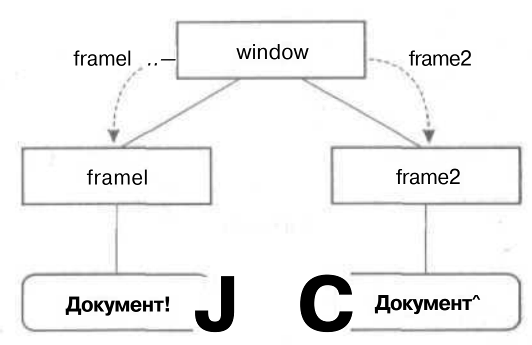


Рис. 2.12. Обращение к фреймам по имени

Иногда может потребоваться получить доступ к родительскому окну. Например, это бывает необходимо, если вы хотите при следующем переходе убрать все фрей-

мы. Удаление фреймов означает лишь загрузку нового документа вместо содержащего фреймы — в рассматриваемом случае загрузку документа в родительское окно. Это можно сделать с помощью доступа к родительскому окну на основе свойства parent (рис. 2.13). Чтобы загрузить новый документ, следует использовать объект location из родительского окна (заметим, что каждый фрейм имеет собственный объект location): требуется внести в location.href родительского окна новый URL-адрес. Таким образом, чтобы загрузить новый документ в родительское окно, требуется записать в сценарии следующее:

parent.location.href="иг1-адрес_документа"

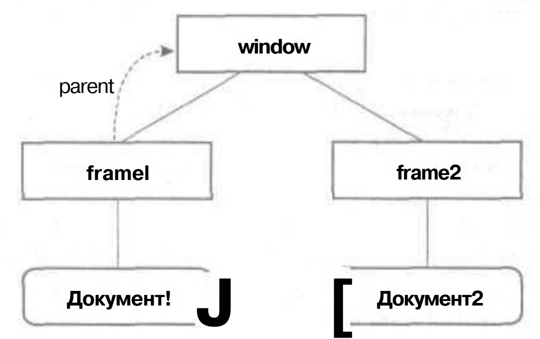


Рис. 2.13. Доступ к родительскому окну на основе свойства parent

Наконец, довольно часто необходимо решать задачу обеспечения доступа из одного фрейма-потомка кдругому (рис. 2.1 4). Например, находясь в одном фрейме-потомке, можно записать что-нибудь в другой фрейм-потомок. Однако между фреймами-потомками не существует прямой связи. Поэтому мы не можем просто вызвать фрейм frarne2 по имени, находясь во фрейме framel. Фрейм framel ничего не знает о frarneZ, но frame2 существует с точки зрения родительского окна. По этойпричинемыдолжнысначалаобратитьсякродительскомуокну, затем — кframe2, и лишь потом к объекту document, который разместился во втором фрейме:

parent. frame2 .document. wri 1e("Привет от первого фрейма! ")

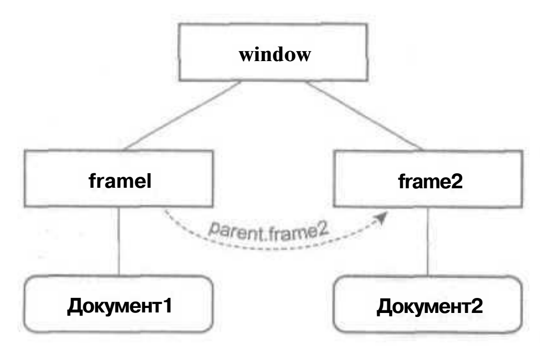


Рис. 2.14. Обеспечение доступа из одного фрейма-потомка к другому

Рассмотрим, как можно изменить элемент в одном фрейме из другого. При щелчке на тексте в правом фрейме в левом изменяется один из текстовых элементов.

Ниже показаны HTML-коды соответственно для левого и правого фреймов, а также порождаемый ими внешний вид страницы (рис. 2.15).

Документ в левом фрейме с именем LEFT:

```
<HTML>
Один<B1?>
Два
<H1 ID = "XXX">Три</H1>
</HTML>
```

Документ в правом фрейме:

```
<HTML>
<5CRIPT>
function changeQ {
parent.LEFT.document.all.XXX.innerText = "Ура!"
!
</SCRIPT>
<H1 onclick = "change0() ">Щелкни здесь</H1>
</HTML>
```

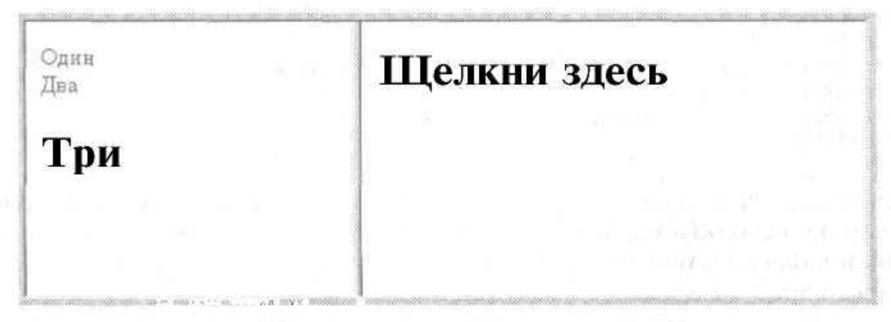


Рис. 2.15. Изменение элемента в одном фрейме из другого

В теле функции с hangeQ, которая делает нужные изменения, происходит обращение к левому фрейму с именем LEFT (задается в установочном HTML-файле) через parent. Далее doc ument.all.XXX обеспечивает доступ к элементу с идентификатором "XXX" (в примере это заголовок 1-го уровня). Здесь all — коллекция всех элементов документа. Собственно изменение элемента происходит за счет присвоения значения свойству innerText (в примере это слово "Ура!").

Обратите внимание, что изменения в одном фрейме по событию в другом происходят без перезагрузки HTML-документа. Для изменения элементов можно использовать, кроме innerText, свойства outerText, innerHTML и outerHTML. О них еще будет рассказано в подразделе 2.6.3. Выбор свойства зависит от того, что именно и насколько вы хотите изменить.

Описанным выше способом вы можете организовать, например, такой сценарий: щелчок на миниатюре (маленьком изображении) в одном фрейме выводит в нем же полномасштабное изображение, а в другом — краткое описание.

Фреймы удобно использовать при создании навигационных панелей. В одном фрейме располагаются ссылки, а второй предназначен для отображения документов, вызываемых при активизации соответствующих ссылок. При активизации ссылки документ загружается не в тот же фрейм, где находятся ссылки, а в другой.

Пример

Навигационная панель. Окно браузера разделено на два фрейма: первый выполняет роль навигационной панели, а второй — окна для отображения документов.

frames. htm - установочный файл.

```
<HTML>
<FRAMESET COLS = "25%,75%">
<FRAME SRC = "menu.htm" NAME = "menu">
<FRAME SRC = "start.htm" NAME = "main">
</FRAMESET>
</HTML>
```

Здесь start.htm —документ, который будет первоначально показан во фрейме main.

menu.htm1 - навигационная панель.

```
<BTME>
<SCRIPT >
function load(url) {
parent.main. location.href = url;
I
</SCRIPT>
<BODY>
<A HREF = "javascript: load('первый.htm')">Первый</A>
<A HREF = "второй.htm" TARGET = "main">Второй </A>
<A HREF = "третий.htm" TARGET = 'Чор">Третий </A>
</BODY>
</HTML>
```

Здесь показано несколько способов загрузки новой страницы во фрейм main. В первой ссылке используется функция load(). Вместо атрибута TARGET указания на фрейм выполняет функция. Функции **loadQ** в качестве параметра передается строка**'первый.htm',** указывающая, какой файл следует загрузить. При этом место, куда он будет загружен, определяется самой функцией load(). Во второй ссылке используется атрибут TARGET. Третья ссылка показывает, как можно избавиться от фреймов. Чтобы удалить фреймы с помощью функции loadQ, достаточно написать **в** ней следующую строку:

```
parent.location.href = url
```

Атрибут TARGET в теге ссылки <A HREF> обычно применяется в тех случаях, когда требуется загрузить одну страницу в один фрейм. Язык сценариев используют, если необходимо при активизации ссылки выполнить несколько действий, например загрузить несколько страниц в разные фреймы.

Для ссылок из родительского окна к объектам его дочерних фреймов можно воспользоваться коллекцией frames всех фреймов. Коллекция фреймов представляет собой массив объектов фреймов. Обратиться к конкретному фрейму из этой коллекции можно по индексу или по имени фрейма, указанному в качестве значения атрибута NAME в теге FRAME

```
window.frames[индекс]
window.имя_фрейма
```

Заметим, что индекс 0 соответствует первому дочернему фрейму в порядке, определенном следованием тегов <FRAME> в контейнере **<FRAMESET>.**

Приведенные выше шаблоны ссылок на фреймы из родительского окна используются как префиксы в полных ссылках к объектам, содержащимся во фреймах.

При этом следует помнить, что если нас интересуют объекты документа, загруженного во фрейм, то прежде чем обратиться к ним, следует упомянуть объект document. Например

```
window.frames(0).document.all.My input.value
window.LEFT.document.all.My input.value
```

Ссылка из дочернего фрейма непосредственно на родительский производится с помощью ключевого слова parent. Если имеется еще один фрейм более высокого уровня, то ссылка на него выглядит так: parent.parent. Аналогичным образом можно построить ссылку из дочернего фрейма до прапрадедушки. Чтобы сразу обратиться к родительскому окну, находящемуся на вершине иерархии, можно использовать ключевое слово top.

СОВЕТ

При использовании ссылки top следует учитывать то обстоятельство, что ваш сайт может быть загружен в один из фреймов другого сайта. В этом случае указанный вами объектtop окажется совсем другим, и ссылки, построенные с его использованием, не будут правильно работать. Поэтому рекомендуется использовать parent для ссылок на вышестоящий фрейм (окно).

Предотвращение использования фреймов

Ссылки top и self можно использовать для предотвращения отображения вашего сайта внутри фреймов другого сайта. Для этого необходимо, чтобы документ верхнего уровня проверял, в какое окно он загружен. Это должно быть самое верхнее (top) или родительское (parent) окно. Если это действительно так, ссылка на свойство top совпадает со ссылкой selfна текущее окно. При несовпадении этих значений документ следует перезагрузить заново, но уже в окно верхнего уровня. Сценарий, выполняющий эту работу, необходимо разместить в начале документа. Вот его код:

```
<SCRIPT>
if (top != self)
   top.location = location
</SCRIPT>
```

Проверка загрузки фреймов

При посещении вашего многофреймового сайта пользователь может сделать закладку (поместить в папку «Избранное») только на один фрейм, в то время как все средства навигации по сайту находятся в другом фрейме. При следующем посещении в.браузер загрузится усеченный вариант вашего сайта. При этом пострадает как пользователь, так и ваша репутация.

Следующий простой сценарий проверяет, загружается ли веб-страница в своем наборе фреймов, сравнивая URL-адреса окна top и текущего окна. Если они совпадают, то необходимо загрузить набор фреймов, определяемый в файле frameset.htm.

```
<SCRIPT>
if  (top.location.href == window.location.href)
   top.location.href  =  "frameset.htm"
</SCRIPT>
```

2.5.3. Плавающие фреймы

Для вставки одного HTML-документа в тело другого средствами браузера пользователя, а не сервера, служит контейнерный тег<IFRAME>:

```
<IFRAME SRC = "адрес_документа"></1PHAME>
```

Элемент, задаваемый этим тегом, можно позиционировать с помощью параметров таблицы стилей (тег <5Tт!E> или атрибут STYLE). Если его положение не указано явно, то он позиционируется в соответствии с положением <IFRAME>B HTML-коде. Внешне этот элемент выглядит как прямоугольная область (с полосами прокрутки или без них), в которой отображается документ из некоторого HTML-файла. Такие окна иногда называют плавающими фреймами.

HTML-документы, загружаемые в плавающие фреймы, могут иметь сценарии и прочие средства, присущие любому HTML-документу. Ниже приведен пример загрузки трех различных HTML-файлов в плавающие фреймы (рис. 2.16). Это HTML-документы из сайта автора (www.admiral.ru/-dunaev).

```
<HTM1 >
<H3>Включение HTML-документов на стороне клиента</H3>
<H4>Использование тега < iframe></H4>
Здесь в трех окнах показаны страницы моего сайта<Ьг>
<IFRAME SRC = ""examples.htm" ></IFRAME>
<IFRAME SRC = "flashx.htm" ></ IFRAME >
<p>
<IFRAME SRC = "i_is.htm" STYLE = "posi tio n: absolute; t op: 310; width:500;
  height:250" SCROLLING = no" ></IFRAME >
</HTML >
```

Другой пример использования плавающих фреймов рассмотрен в подразделе 4.2.2 (глава 4).

Плавающий фрейм ведет себя почти так же, как и обычный фрейм. Если он создан, то его можно найти в коллекции frames. Среди его свойств рассмотрим свойство align — выравнивание плавающего фрейма относительно окружающего содержимого документа. Возможные значения:

- absbottom — выравниваетнижнюю границу фреймаподстрочнойлинии символов окружающего текста;
- absmiddle — выравнивает середину фрейма по центральной линии между top и absbottomокружающеготекста;
- baseline — выравнивает нижнюю границу фрейма по базовой линии окружающего текста;
- bottom — совпадает с baseline (только в IE);
- left — выравнивает фрейм по левому краю элемента-контейнера;
- middle — выравнивает воображаемую центральную линию окружающего текста по воображаемой центральной линии фрейма;
- right — выравнивает фрейм по правому краю элемента-контейнера;
- texttop — выравнивает верхнюю границу фрейма по надстрочной линии символов окружающего текста;
- top — выравнивает верхнюю границу фрейма по верхней границе окружающего текста.

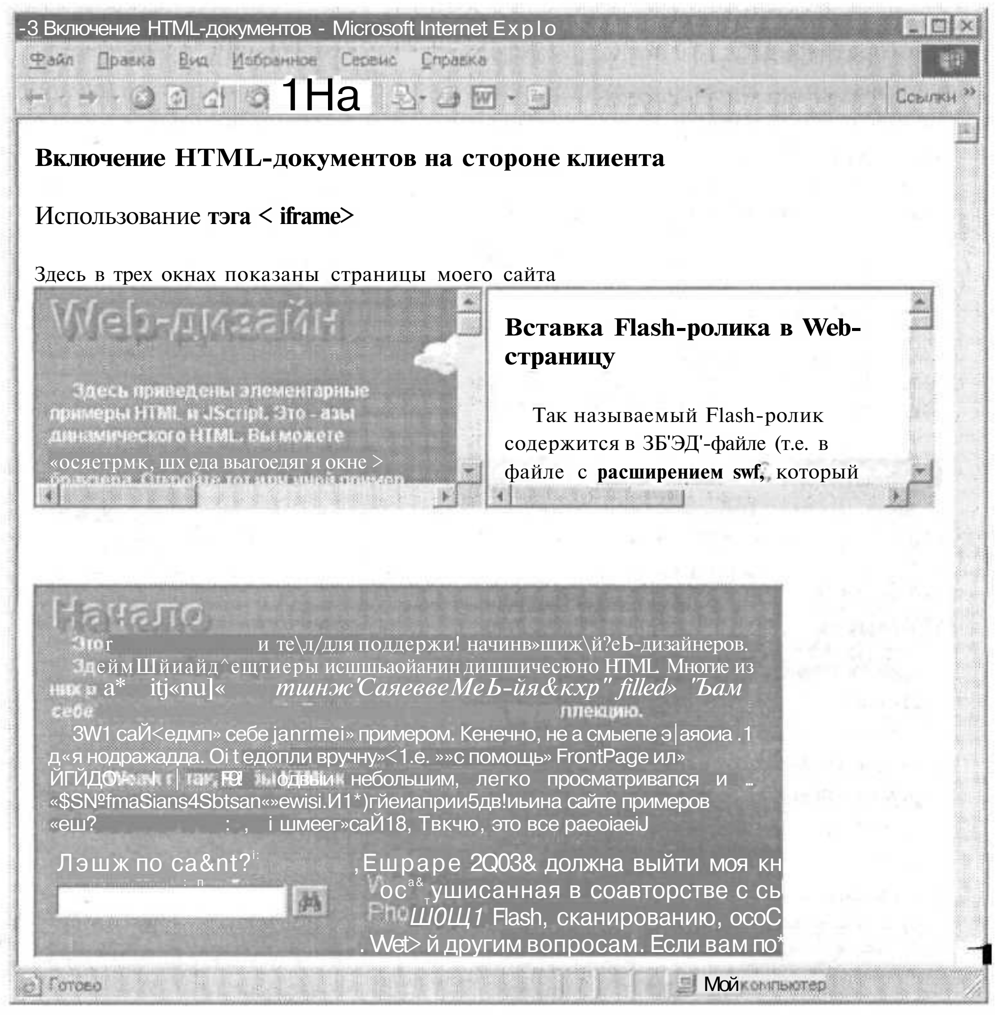


Рис. 2.16. Документ с тремя документами из внешних файлов, загруженными в плавающие фреймы

2.5.4. Всплывающие окна

Всплывающее окно не имеет органов управления и располагается над документом, в котором оно было создано, в том числе и над диалоговыми окнами. Щелчок кнопкой мыши где-либо вне этого окна или вызов другого приложения приводит к его закрытию. Оно также не отображается на панели задач Windows. После создания всплывающего окна его заполняют содержимым. При этом необходимо следить, чтобы содержимое вписывалось в заданные границы окна, поскольку оно не имеет полос прокрутки. Такие окна удобно использовать в качестве больших всплывающих подсказок для вывода контекстной справочной информации.

Всплывающим окнам соответствует объект popup. Он создается только в сценарии для браузера IE5.5+ с помощью метода window.createPopup().

Ниже приводится пример типичной последовательности выражений сценария, создающего всплывающее окно, задающего его параметры и показывающего его на экране (рис. 2.17):

```
var mypopup = window.createPopup()             // создание всплывающего окна
var popupBody = mypopup.document.body
/* Параметры окна */
popupBody.style.border = "solid 1px green"     // граница окна
popupBody.style.padding = "5px"                // отступ текста
popupBody.style.color = "green"                // цвет текста в окне
popupBody.style.background = "ffffd0"          // цвет фона окна
/* Заполнение содержимым: */
popupBody.innerHTML = "<H3>Здесь расположен некоторый текст</H3>"
mypopup.show(200,100,400,70,document.body)    // вывод всплывающего окна
```

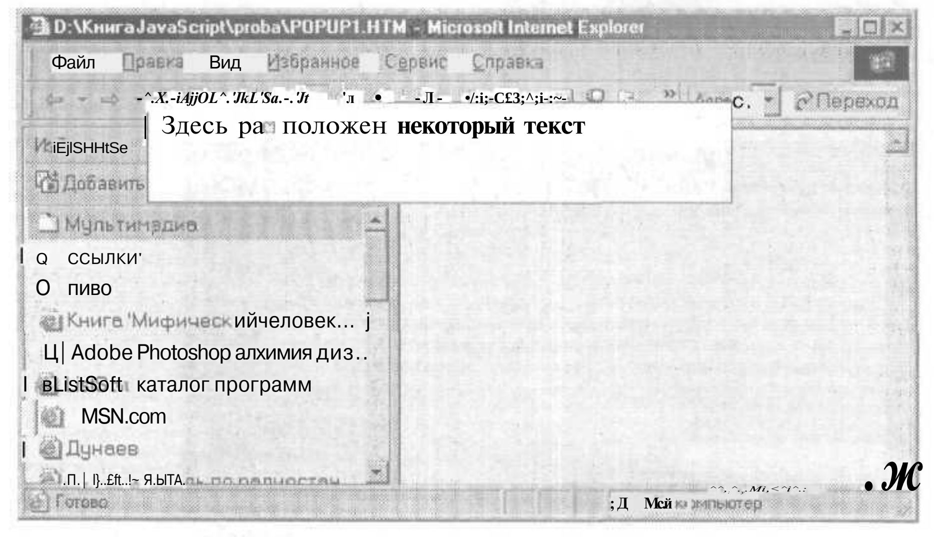


Рис. 2.17. Всплывающее окно на фоне браузера

Свойства объекта popup:

- document — свойство, имеющее в качестве значения ссылку на документ, содержащийся во всплывающем окне; через него можно задать ряд параметров как самого окна, так и его содержимого:

```
mypopup.document.body.style.border = "solid 4px black"   // границы окна
mypopup.document.body.style.background = "yellow"        // цвет фона окна
mypopup.document.body.style.color = "blue"               // цвет текста в окне
```

- isOpen — пока всплывающее окно отображается на экране, это свойство имеет значение true, в противном случае — false; применяется в выражениях сценария, которые выполняются при уже открытом окне.

Методы объекта popup:

- show(left, top, width, height [, позиционирование]) — отображает всплывающее окно после создания объекта popup с помощью метода window.createPopup()

и заполнения его содержимым. Если окно исчезло из-за того, что пользователь щелкнул где-то в поле браузера, то для его повторного открытия следует заново выполнить метод show(). Первые четыре параметра метода задают координатное позиционирование и размеры окна. Параметры left и top определяют координаты верхнего левого угла окна относительно экрана монитора, а не окна браузера. Последний необязательный параметр позволяет задать другое координатное пространство как ссылку на элемент HTML-документа. Например, указав document.body, вы ограничиваете координатное пространство окном браузера.

- hideQ — позволяет скрыть созданное и отображаемое всплывающее окно.

Заполнение всплывающего окна содержимым производится с помощью свойства innerHTML, принимающего в качестве значения строку, содержащую теги HTML. Более подробно это свойство будет рассмотрено в подразделе 2.6.3. Таким образом, в окно можно вставлять не только тексты, но и изображения и другие элементы.

Пример

```
mypopup.document.body.innerHTML = "<IMG SRC = 'picture.jpg'>"
```

Более подробно об этом свойстве будет рассказано в следующем разделе.

Ниже приводится пример HTML-документа (рис. 2. 18) с двумя кнопками, щелчок на которых вызывает одну и ту же функцию pop(xcontent), открывающую всплывающее окно и заполняющую его содержимым, которое задано параметром xcontent:

```
<HTML>
<H3>Всплывающие окна</H3>
<BUTTON onclick = "pop('Во первых строках своего письма я рад вам
сообщить ')">Просто текст</B11TTOM>
<BR><BR>
<BUTTON onclick = "pop('<IMG SRC = кружка.gif>
Это кружка с квасом. ') ">Картинка с текстом</B1)TTOM>
<SCRIPT>
function pop(xcontent){
var mypopup = window.createPopup();
var popupBody = mypopup.document.body
popupBody.style.border = "solid 2px green"
popupBody.style.padding = "5px"
popupBody.style.color = "blue"
popupBody.style.background = "ffffd0"
popupBody.innerHTML = "<p>" + xcontent + "</p>"
mypopup.show(130,90,400,200,document.body)
]
</SCRIPT>
</HTML>
```

В рассмотренном выше примере при вызове функции pop() каждый раз создается одно и то же всплывающее окно, содержимое которого может изменяться. Поскольку позиционирование, размеры и цвета окна не изменяются, то можно написать более экономный код за счет вынесения выражений его создания и параметризации за пределы определения функции pop():

```
<HTML>
<H3>Всплывающие окна</H3>
<BUTTON onclick = "pop('Во первых строках своего письма я рад вам
```

```
сообщить ') ">Просто текст</BUTTON>
<BR><BR>
<BUTTON onclick = "pop('<IMG SRC = кружка.gif>
Это кружка с квасом. ') ">Картинка с текстом</BUTTON>
<SCRIPT>
var mypopup = window.createPopup()
var popupBody = mypopup.document.body
popupBody.style.border = "solid 2px green"
popupBody.style.padding = "5px"
popupBody.style.color = "blue"
popupBody.style.background = "ffffd0"

function pop(xcontent) {
popupBody.innerHTML = "<p>" + xcontent + "</p>"
mypopup.show(130,90,400,200,document.body)
}
</SCRIPT>
</HTML>
```

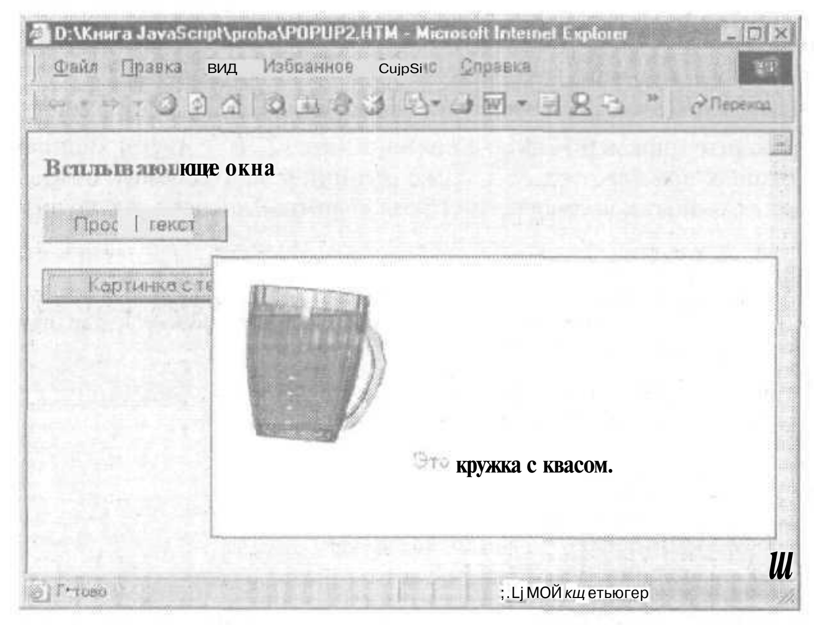


Рис. 2.18. Пример всплывающего окна с изображением и текстом

2.6. Динамическое изменение элементов документа

HTML-документ, загруженный в браузер, можно изменять с помощью сценариев. Поскольку речь идет не о файле на диске, а о его образе в браузере в оперативной памяти компьютера, то говорят о динамическом изменении документа. Существует три основных способа динамических изменений:

* с помощью метода write();

- путем изменения значений свойств, соответствующих атрибутам HTML-тегов и параметрам каскадных таблиц стилей;
- путем изменения значений свойств innerText, outerText, innerHTML,outerHTML, которые имеют почти все объекты, заданные с помощью тегов.

2.6.1. Использование метода writeQ

Метод writeQ объекта document уже неоднократно упоминался в этой книге. Он принимает в качестве параметра строку, содержащую HTML-код и/или просто текст. Выполнение в сценарии выражения document.write(сТроКа) приводит к дописыванию в текущий HTML-документ содержимого параметра строка и немедленной его интерпретации браузером. В результате документ и его объектная модель обновляются. При этом файл с исходным HTML-кодом остается без изменений. Если требуется полностью заменить текущий документ, то сначала применяют метод очистки документа document.dearQ, а затем document.write(сТроКа). Однако при такой кардинальной трансформации текущего документа следует быть осторожным. Наиболее безопасный прием — сначала сгенерировать содержимое нового HTML-документа с помощью сценария в текущем документе, а затем отправить HTML-код в новое окно или в другой фрейм многофреймового документа.

Если быть более точным, то следует отметить, что метод writeQ может принимать произвольное количество строковых параметров:

```
document. wri te CcrpOKal, [, строка2 . . .[, строкам]])
```

Здесь квадратные скобки указывают лишь на необязательность заключенных в них параметров. Если указывается несколько параметров, то они разделяются запятыми. Заметим также, что весь HTML-код документа можно записать как одну строку.

Кроме метода writeQ можно использовать для тех же целей и метод writelnQ, имеющий такой же синтаксис. Его особенность в том, что он добавляет в конце каждой строки документа невидимый символ перехода на другую строку.

Методы writeQ и writelnQ работают в браузерах IE3+ и NN2+.

2.6.2. Изменениезначений атрибутов элементов

Элементы HTML-документа, как известно, задаются тегами, большинство из которых имеют атрибуты (параметры). В объектной модели документа тегам элементов соответствуют объекты, а атрибутам — свойства этих объектов. В большинстве случаев названия свойств объектов совпадают с названиями атрибутов, но в отличие от последних записываются в нижнем регистре. Это же относится и к параметрам таблиц стилей. Однако это общее правило может иметь исключения. Поэтому им нужно пользоваться так: ищите в справочнике свойство, похожее на атрибут, и обращайте внимание на его написание. Например, тегу графического изображения <IMG ID = "myimg" SRC = "picture.jpg"> соответствует объект document.all.myimg, а атрибуту SRC — свойство document.allmyimg.src, значением которого является имя (URL-адрес) файла с изображением. С помощью сценария можно присвоить этому свойству новое значение, и в HTML-документе произойдет замена графического элемента.

Многие параметры элементов задаются с помощью таблиц стилей, например посредством атрибута STYLE. Так, для позиционирования изображения можно использовать следующий HTML-код:

```
<IMG ID = "myimg" SRC = "picture.jpg"
STYLE = "position:absolute; top:20; left: 50: z-index:3">
```

Чтобы изменить в сценарии параметры стиля элемента, следует присвоить новые значения соответствующим свойствам объекта style:

```
document.all.myimg.style.top = 30
document.all.myimg.style.top = 100
document.all.myimg.style.zIndex = -2
```

Параметр z-index в таблице стилей, указывающий слой (относительное положение выше-ниже) для элемента, в объектной модели представляется свойством zIndex.

ВНИМАНИЕ

Между обозначениями в HTML и таблицах стилей, с одной стороны, и обозначениями соответствующих свойств объектов, с другой стороны, много общего, но имеются и различия. Поэтому будьте внимательны!

Многочисленные примеры использования данного способа изменения параметров элементов документа рассматриваются в главе 4, в частности в разделе 4.1.

2.6.3. Изменение элементов

Наиболее удобный способ динамического изменения элементов HTML-документа в IE4+ основан на использовании свойств innerText, outerText, innerHTML и outerHTML В NN6+ можно использовать только свойство innerHTML. Множество примеров вы найдете в главе 4, а здесь мы ограничимся разъяснением общих вопросов.

Теги элементов документа могут содержать текст и/или другие теги. С помощью перечисленных выше свойств можно получить доступ к содержимому элемента. Изменяя значения этих свойств, можно изменить сам элемент, частично или полностью. Например, можно заменить только надпись на кнопке, а можно превратить кнопку в изображение или Flash-анимацию.

Значением свойства innerText является все текстовое содержимое, заключенное между открывающим и закрывающим тегами элемента. Если в эту строку входят HTML-теги, то они игнорируются. Обратите внимание, что в значение свойства innerText данные открывающего и закрывающего тегов соответствующего элемента не входят. Таким образом, innerText следует понимать как весь внутренний текст, содержащийся в контейнере.

Свойство outerText аналогично свойству innerText, но отличается от него тем, что включает в себя и данные, содержащиеся в открывающем и закрывающем тегах элемента. Таким образом, outerText следует понимать как весь текст, содержащийся в контейнере, включая и его внешние теги.

Рассмотрим в качестве примера следующий фрагмент HTML-кода, выводящий ссылку, изображение и форматированный текст:

```
<DIV ID = "my">
<A HREF = 'raznoe.htm'>
<IMG SRC = ''picture.jpg'>Ссылка на раздел <B>Разное</B>
</A>
</DIV>
```

В этом случае свойства innerText и outerText для элемента, заданного контейнерным тегом <DIV>, совпадают:

```
document.all.my.innerText    // значение равно: "Ссылка на раздел
Разное"
```

При присвоении свойствам innerText и outerText новых значений следует иметь в виду, что если значение содержит HTML-теги, то они не интерпретируются, а выводятся на экран просто как текст. Так, для приведенного выше фрагмента **HTML**-кода выполнение в сценарии выражения

```
document.all.my.innerText = "<ЕШТТОМ>Щелкни здесь</В11ТТОМ>"
```

не создаст в документе кнопку, а лишь выведет строку с текстом, указанным справа от оператора присвоения.

Рассмотренные выше свойства innerText и outerText не столь эффективны, как замечательные свойства innerHTML и outerHTML. Без них мне было бы просто скучно заниматься программированием на HTML и JavaScript.

Свойство innerHTML для любого элемента имеет в качестве значения строку, содержащую HTML-код, заключенный между открывающим и закрывающим тегами элемента. Иначе говоря, innerHTML содержит внутренний HTML-код контейнера элемента. Присвоение этому свойству нового значения, содержащего HTML-код, приводит к интерпретации этого кода. Разумеется, новое значение может и не содержать тегов.

Свойство outerHTML аналогично свойству innerHTML, но отличается тем, что содержит весь HTML-код, включая внешние открывающий и закрывающий теги элемента.

Для приведенного выше фрагмента HTML-кода имеет место следующее:

```
document.all.my.innerHTML    /* значение равно:
                                "<A HREF = 'raznoe.htm'>
                                <IMG SRC = 'picture.jpg'>
                                Ссылка на раздел <B>Разное</B></A>" */
document.all.my.outerHTML    /* значение равно:
                                "<DIV ID = 'my' >
                                <A HREF = 'raznoe.htm'>
                                <IMG SRC = 'picture.jpg'>
                                Ссылка на раздел <B>Разное</B>
                                </A></DIV>" */
```

Если в сценарии выполнить, например, выражение:

```
document.all.my.innerHTML = "<BиTTOM>Щелкни здесь</B1ЯTOM>",
```

то ссылка, изображение и текст будут заменены кнопкой с надписью Щелкни здесь. При этом контейнерный тег <DIV ID = "my"> сохранится. Если же вместо свойства innerHTML использовать свойство outerHTML, то кнопка также появится, но уже без контейнера <DIV ID = "my">.

Заметим, что свойства innerHTML и outerHTML могут применяться и к элементам, которые задаются неконтейнерными тегами, например тегом <IMG>. В этом случае значения innerHTML и outerHTML всегда совпадают.

Ниже приводится код функции getproperties (xid), которая может использоваться в сценарии для получения значений свойств innerHTML, outerHTML, innerText и outerText элементов HTML-документа. Эта функция принимает в качестве параметра значение идентификатора ID элемента (или значение его атрибута NAME) и выводит диалоговое окно со значениями перечисленных выше свойств. Вы можете применять ее на этапе изучения.

```
function getproperties(xid) {
var x = eval ( "doc"document.all." + xid)
alert(" innerHTML: " + x. innerHTML + "\n outerHTML: " + x.ou TERHTML +
"\n innerText: " + x.ii :rText + "\n outerText: " + x.outerText)
}
```

Вот пример использования функции getproperties(xid):

```
<HTML>
<DIV ID = "my">
<A HREF = 'raznoe.htm'>
<IMG SRC = 'picture.jpg' >Ссылка на раздел <B>Разное</B></A>
  </DIV>
  <H2 ID = "pri vet">ripHBeT <1>читателям</1></H3»
  <SCRIPT>
  getproperties("my")                   // отображение свойств элемента my
  getproperties ("privet")              // отображение свойств элемента
privet

  function getproperties("xid"){
        // код функции
  }
  </SCRIPT>
</HTML>
```

2.7. Загрузка изображений

Большинство веб-страниц содержат графические элементы, которые используются не только для украшения страниц, но и в качестве информационного наполнения: иллюстрированные каталоги товаров, схемы, чертежи, географические карты, фотогалереи и т. п. Если файлы с графикой велики и/или их много, то загрузка такой страницы в браузер может потребовать слишком много времени. Для ускорения загрузки используют специальные приемы. Так, нередко сначала загружают изображения с низким разрешением из небольших по объему файлов. На их основе создаются ссылки на файлы с графикой полноценного разрешения, которые загружаются только при щелчке на ссылке. Можно также использовать в теге <IMG> кроме атрибута SRC атрибут LOWSRC, позволяющий загрузить сначала изображение с низким разрешением, а затем, по мере приема, заменить ее рисунком с большим разрешением. Здесь мы рассмотрим способ ускорения загрузки графики, предоставляемый JavaScript.

С помощью сценария можно организовать предварительную загрузку изображений в кэш-память браузера, не отображая их на экране. Это особенно эффективно при начальной загрузке страницы. Пока изображения загружаются в память, оставаясь невидимыми, пользователь может рассматривать текстовую информацию, его не раздражает медленное появление графических элементов. В результате загрузка страницы не вызывает у пользователя отрицательных эмоций.

Для предварительной загрузки изображения следует создать его объект в памяти браузера. Такой объект несколько отличается от объекта изображения, создаваемого с помощью тега <IMG>. Как и все объекты, создаваемые сценариями, объект изображения не отображается в окне браузера. Однако его наличие в коде документа уже обеспечивает загрузку самого изображения при загрузке документа. Чтобы создать в памяти объект изображения, необходимо в сценарии выполнить следующее выражение:

```
myimg = new 1ma§e(ширина, высота)
```

Параметры функции-конструктора объекта определяют размеры изображения и должны соответствовать значениям атрибутов WIDTH и HEIGHT тега <IMG>, который используется для отображения предварительно загруженного изображения.

Для созданного в памяти объекта изображения myimg можно задать имя или, в общем случае, URL-адрес графического файла. Это делается с помощью свойства src:

```
myimg. src = "11K1_-адрес_изображения"
```

Данное выражение предписывает браузеру загрузить в кэш-память указанное изображение, но не отображать его. После загрузки в кэш-память всех изображений и загрузки всего документа можно сделать их видимыми. Для этого нужно свойству src элемента <IMG> присвоить значение этого же свойства объекта изображения в кэш-памяти. Например:

```
document.images[0].src = myimg.src
```

Здесь слева от оператора присвоения указано свойство src первого в документе элемента, соответствующего тегу <IMG>, а справа — свойство src объекта изображения в кэш-памяти.

Теперь рассмотрим в качестве примера HTML-документ, в котором отображается список названий графических элементов и одно исходное изображение (рис. 2.19). Щелчок на элементе списка приводит к отображению соответствующего изображения. Все графические элементы из этого списка предварительно загружаются в кэш-память и поэтому быстро отображаются при выборе из списка. Список, как известно, создается с помощью контейнерного тега <SELECT>, содержащего теги <OPTION>. Соответствующий фрагмент HTML-кода мы сгенерируем с помощью сценария и запишем в текущий документ. В листинге 2.3 приведен соответствующий код.

Листинг 2.3. Код HTML-документа, в котором отображается список названий графических элементов и одно исходное изображение

```
<HTML>
<HEAD>
<SCRIPT>
var imgFile = new ArrayO                    // массив имен графических
                                            // файлов
imgFile[0] = "pictl.jpg"
imgFiletl] = "pict2.jpg"
imgFile[2] = "pict3.jpg"
imgFile[3] = "pict4.jpg"

var imgName = new ArrayO                    // массив названий картинок
imgName[0] = "Картинка!"
imgName[l] = "Картинка2"
imgName[2] = "КартинкаЗ"
```

```
imgName[3] = "Картинка4"

/* Создание объектов изображений и загрузка изображений в кэш */
var imgObj = new ArrayO                    // массив объектов изображений
for(i = 0; i < imgFile."length; i-
imgObj[i] = new Image(150, 100)            // создаем объект изображения
imgObj[i].src = imgFile[i]                 /* загрузка рисунка в память
                                              без отображения */
}

function imgshowClist) {                   /* отображение изображения
                                              при выборе из списка
var x = list.options[list. selectedIndex].value   /* выбранный номер
                                                     изображения */
document.all.imgO.src = eval("imgObj [ "+ x + "].src")
}

/* Создание списка изображений */
var clist = "<SELECTonchange= ' imgshow(this) ' >"
for(i = 0; i<imgFile. length; i++){
clist+= "<OPTION VALUE=" + i + "">" + imgName[i]
}
clist+= "</5ELECT>"
document.write(clist)                      /* запись списка изображений
                                              в документ */
</SCRIPT>
</HEAD>
<! Исходная картинка>
<IMG ID = "img0" SRC = "pict0.jpg" WIDTH=150 HEIGHT=100>
</HTML>
```

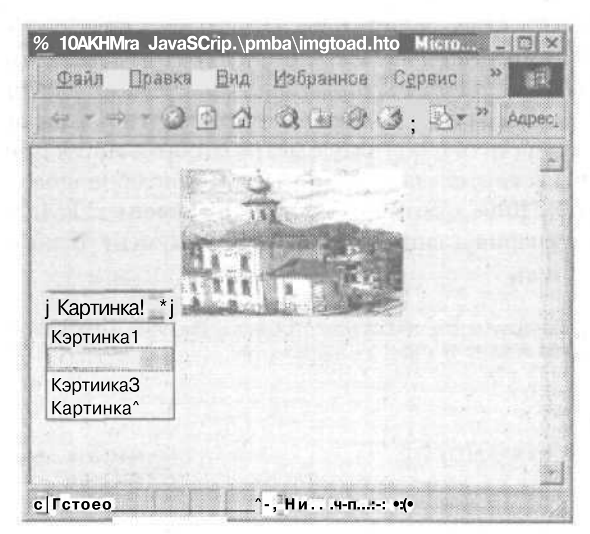


Рис. 2.19. При выборе названия изображения из списка соответствующее отображается из предварительно загруженного файла

Обратите внимание, что для преобразования строки в настоящую ссылку на объект используется функция evalQ. HTML-код, определяющий список изображений и сгенерированный сценарием, выглядит следующим образом:

```
<SELECT onchange = .' imgshow(this) ' >
«DPTION VALUE = бЖартинка!
<OPTION VALUE = 1>Картинка2
<<OPTION VALUE = 2>Картинка3
<<OPTION VALUE = 3>Картинка4
</ 5ELECT>
```

2.8. Управление процессами во времени

Вы можете периодически, через заданный интервал времени, запускать код (например, функцию) JavaScript. При этом создается эффект одновременного (параллельного) выполнения вычислительных процессов. Например, вы можете запустить несколько функций, перемещающих на экране различные видимые объекты. Эти объекты будут двигаться как бы одновременно. Иногда требуется организовать временную задержку перед выполнением какой-то функции, чтобы ранее начатый процесс успел завершиться. Все это относится к задачам управления вычислительными процессами во времени.

Для организации постоянного периодического (через заданный интервал времени) выполнения некоторого выражения или функции служит метод **setlntervalQ** объекта window. Этот метод имеет следующий синтаксис:

```
setlnterval (выражение, период [, язык])
```

Первый параметр представляет собой строку, содержащую выражение (в частности, вызов функции). Второй параметр — целое число, указывающее временную задержку в миллисекундах перед последующими выполнениями выражения, указанного в первом параметре. Третий, необязательный параметр указывает язык, на котором написано выражение; по умолчанию — JavaScript. Метод setlntervalQ возвращает некоторое целое число — идентификатор временного интервала, который может быть использован в дальнейшем для прекращения выполнения процесса, запущенного с помощью данного метода (см. ниже метод **clearlnterval()**).

Пусть, например, требуется, чтобы некоторая функция myfuncQвыполнялась периодически через 0,5 с. Тогда в сценарии следует записать следующее выражение:

```
setlnterval("myfunc()", 500)
```

Тот факт, что первый параметр метода setlntervalQ является строкой, обусловливает некоторые особенности передачи параметров периодически вызываемой функции. Если периодически вызываемая функция принимает параметры, то мы должны сначала сформировать строку, содержащую имя этой функции, круглые скобки, значения параметров и запятые между ними, а затем передать ее в качестве первого параметра методу setlntervalQ. В следующем примере показано, как передать методу setlntervalQ функцию с двумя параметрами, paraml и param2, значения которых определены в другом месте сценария:

```
van xstr = "myfunc(" + paraml + "," + param2 +")"
setlnterval(xstr, 500)
```

Выражение, переданное методу setlntervalQ, будет периодически выполняться сколь угодно долго. Если это выражение осуществляет, например, приращение координат какого-нибудь видимого элемента документа, то этот элемент будет перемещаться в окне браузера.

Для остановки запущенного временного процесса служит метод clear!nterval(MfleH-тификатор), который принимает в качестве параметра целочисленный идентификатор, возвращаемый соответствующим методом setlntervalQ, например:

```
var myproc = setInterval("myfunc() , 100")
if (confirm("Прервать процесс ?"))
   clearInterval(myproc)
```

Другие, содержательные примеры использования методов setlntervalQ и clear-IntervalQ приведены в разделах 4.1 и 4.2.

Чтобы выполнить выражение с некоторой временной задержкой, используется метод setTimeoutQ. Этот метод объекта window имеет следующий синтаксис:

```
setTimeout(выражение, задержка [, язык])
```

Первый параметр представляет собой строку, содержащую выражение (в частности, вызов функции). Второй параметр — целое число, указывающее временную задержку в миллисекундах выполнения выражения, указанного в первом параметре. Третий, необязательный параметр указывает язык, на котором написано выражение; по умолчанию — JavaScript. Метод setTimeout() возвращает некоторое целое число — идентификатор временного интервала, который может быть использован в дальнейшем для отмены задержки выполнения процесса, запущенного с помощью данного метода (см. ниже метод clearTimeoutQ).

Пусть, например, требуется, чтобы некоторая функция myfuncQ выполнялась спустя 1 с. Тогда в сценарии следует записать следующее выражение:

```
setTimeout("myfunc()", 1000)
```

ВНИМАНИЕ

> Помните, что это выражение не задерживает выполнение всех последующих выражений сценария. Оно лишь задерживает выполнение функции myfuncQ.

Для отмены задержки процесса, запущенного с помощью метода setTimeout(), используется метод clearT1teoи1(идентификатор), который принимает в качестве параметра целочисленный идентификатор, возвращаемый соответствующим методом setTimeout.

В следующем HTML-документе имеются две кнопки. Щелчок на кнопке Пуск открывает через 5 с новое окно и загружает в него документ mypage.htm. Однако это действие можно отменить с помощью кнопки Отмена, если щелкнуть на ней, пока окно еще не открыто:

```
<HTML>
«BUTTON ID="start">Пуск</BuTTOM>
«BUTTON ID="stop">OTMeHa</BUTTON>
<5CRIPT>
var myproc
function start.onclickO{
myproc = setTimeout ("window.open('mypage.htm')", 5000)
}

function stop.onclickO{
clearTimeout(myproc)
}
</SCRIPT>
</HTML>
```

Более содержательный пример использования метода setTimeoutQ можно найти в подразделе4.8.10.

ВНИМАНИЕ

При использовании методов setIntervalQ и setTimeout() следует иметь в виду, что их вторые параметры задают лишь приблизительные значения временных задержек.

2.9. Работа с Cookie

Для хранения небольших объемов информации на диске компьютера пользователя в браузере предусмотрен так называемый механизм cookie. Обычно он используется для хранения имени пользователя и пароля, который вводится в поле формы защищенного веб-сайта, атакже информации о предыдущем посещении сайта. Например, можно сохранить на диске дату последнего посещения сайта данным пользователем. При загрузке сайта эта дата сравнивается с некоторой датой, установленной автором сайта в качестве даты обновления. Если вторая (авторская) дата более поздняя, чем первая, то на веб-странице появляется соответствующая отметка, например текст или изображение с надписью «New» или «Обновлено!». Разумеется, сравнение дат и вывод на страницу отметки производится с помощью сценария. А вот еще один пример: сценарий проверяет, а посетил ли сайт, скажем, Вася Пупкин; если это произошло, то на страницу выводится персональное сообщение.

По существу, cookie — это единственный способ сохранения данных на диске пользователя, безопасный для него. Как известно, веб-браузеры препятствуют свободному обращению к папкам и файлам на компьютере пользователя (см. главу 5). Однако следует помнить, что многие пользователи не любят cookie-записи и всячески их истребляют.

Записи cookie браузер Internet Explorer сохраняет в отдельных текстовых файлах, расположенныхв папке Windows\Cookies. Имя такого файла образуется на основе имени пользователя и домена того сервера, на котором создавался cookie-файл. Netscape Navigator 4 для Windows создает просто один файл cookie.txt. Вообще говоря, структура данных в cookie-файлах для различных браузеров не столь существенна, поскольку, во-первых, не рекомендуется открывать и изменять эти файлы в текстовых редакторах, а во-вторых, браузеры IE И N N используют одинаковый синтаксис чтения и записи cookie-данных, основанный на использовании свойстваdocument. cookie.

Итак, данные в cookie-файлах организованы в виде записей. Каждую такую запись можно представить себе в виде строки, содержащей следующие элементы:

- имя записи;
- содержание записи;
- срок хранения (годности) записи;
- домен сервера, который создал запись;
- сведения о необходимости установки безопасного http-соединения для доступа к записям;

- расположение документов, которым разрешен доступ к записям.

Зависимость записей от домена обеспечивает безопасность хранения так называемых невосстанавливаемых паролей (пар вида имя_пользователя-пароль), поскольку запись, созданную сервером одного домена, не может прочитать сервер с другим доменом.

Срок хранения используется браузером для автоматического удаления просроченных записей, чтобы предотвратить чрезмерное разрастание объема cookie-файлов. Впрочем, IE4+ и NN4+ ограничили объем cookie-файлов 20 записями на каждый домен.

Для записи данных в cookie-файл с помощью JavaScript используется выражение присвоения строки, содержащей cookie-данные, свойству document.cookie. При этом важно соблюдать формат строки:

```
document.cookie = "coo!<1eMaтe=данные
 [; expires=СТроКа_ВреМеНК_GMT] '
 [; path=nyTb]
 [; domain=домен]
 [; secure]"
```

Здесь квадратные скобки указывают, что заключенное в них содержимое не является обязательным (может быть опущено).

Рассмотрим элементы cookie-строки.

- *Им я-данные.* Каждая cookie-запись должна иметь имя и строковое значение, которое может быть и пустой строкой. Например, если требуется сохранить слово «Вася» в cookie-записи с именем User_Name, то соответствующее выражение JavaScript будет иметь вид:

  ```
  document.cookie = "User_Name=Bacfl"
  ```

  При выполнении этого выражения браузер пытается найти cookie-запись с таким именем. Если он не находит ее в текущем домене, то создает ее автоматически. Еслin запись с таким именем уже существует, то браузер заменяет ее данные новыми. Данные в этом элементе cookie-записи не должны содержать точек с запятыми, запятых и пробелов. Чтобы заменить пробелы соответствующими символами (%20),), строка с данными предварительно обрабатывается функцией escapeQ.

- *Срок хранения.* Дата и время хранения (годности) cookie-записи должны быть представлены строкой и содержать данные по Гринвичу (GMT). Например, вычислить дату истечения срока хранения записи месяц спустя после текущей даты можно следующим образом:

  ```
  var expdate = new Date()                    // создаем объект даты
  var monthFromNow = expdate . getTime() + (30*24*60*60*1000)
  expdate.setTime(monthFromNow)                 // устанавливаем значение даты
  ```

После этого полученную дату следует привести к строковому формату GMT:

```
document.cookie = "User_Name=BacH; expires=" + expdate. toGMTString()
```

Cookie-запись можно удалить и до истечения заданного срока хранения, установив новый срок, заведомо уже прошедший:

```
expdate=Thu. 01-Jan-70 00:00:01 GMT
```

Отсутствие срока хранения означает для браузера, что данная cookie-запись является временной и не записывается в файл.

Об использовании методов объекта даты см. подраздел 1.7.5.

- path — cookie-записи, производимые компьютером пользователя, имеют путь, принятый по умолчанию (в текущей папке). Однако можно создать копию cookie в другой папке, указав путь к ней в качестве значения этого параметра.
- domain — для синхронизации cookie-данных с определенным документом или и группой документов браузер выясняет домен текущего документа и помещает в cookie-файл записи, соответствующие этому домену. Если пользователю требуется просмотреть список всех cookie-записей, содержащихся в свойстве **document.cookie,** то он должен просмотреть все пары имя-значение, находящиеся в cookie-файле с именем домена текущего документа. Формат представления домена должен охватывать по крайней мере два уровня, например rambler.ru.
- secure — принимает логические значения (true или false). При создании cookie-записей на стороне клиента (компьютера пользователя) этот параметр опускается.

Теперь займемся чтением и записью данных cookie. Данные cookie, которые можно получить с помощью сценария на JavaScript, представляют собой единственную строку — значение свойства document. c o okie. Выбор значений отдельных элементов (параметров) cookie производится на основе анализа содержимого этой строки методами объекта String (см. подраздел 1.7.1). Кроме того, если две и более cookie-записи (до 20) соответствуют одному и тому же домену, то в JavaScript они все равно представляются одной строкой и разграничиваются точкой с запятой и пробелом.

Рассмотрим функцию readCookie(name), читающую cookie-данные, соответствующие имени записи name, которое передается этой функции в качестве параметра:

```
function readCookie(name) {              // чтение cookie-данных записи
var xname = name + "="
var xlen = xname.length
var den = document.cookie.length
var i = O
while(i < clen) {
var j = i + xlen
if (document.cookie.substri ng(i , j) == xname)
      return getCookieVal(j)
i = document .cookie. indexOf(" ",!) +1
if (i == 0) break
I
return null
I
```

Функция readCookie(name) возвращает значение cookie-записи с именем name или null, если такая запись не найдена. В теле этой функции использована еще одна, вспомогательная функция getCookieVal(j), возвращающая декодированное значение cookie-данных. Декодирование производится с помощью встроенной функции **unescapeQ.** Дело в том, что cookie-запись должна представлять собой кодированную строку, полученную путем обработки встроенной функцией **escapeQ,**

чтобы, в частности, заменить пробелы специальными символами (%20). Код этой функции приведен ниже:

```
function getCookieVal (n) {           /* вспомогательная функция,
                                         вызываемая из readCookieQ */
var endstr = document. cookie. indexOf (";", n)
if (endstr == -1)
   endstr = document .cookie .length
return unescape(document.cookie.substring(n, endstr))
}
```

Теперь рассмотрим функцию **writeCookieQ,** позволяющую создать или обновить cookie-запись:

```
function wri teCookie(name, value, expires, path, domain, secure) {
                                       /* запись cookie */
   document.cookie =
   name + "=" + escape(value) +
   ((expires) ? "; expires=" + expires.toGHTString(): "") +
   ((path) ? "; path=" + path : "") +
   ((domain) ? "; domain=" + domain : "") +
   ((secure) ? "; secure" : "")
   }
```

Эта функция принимает следующие параметры:

- name — строка, содержащая имя cookie-записи (обязательный параметр);
- value — строка, содержащая значение cookie (обязательный параметр);
- expires — объект даты (Date), содержащий срок хранения cookie-записи; если отсутствует, то после завершения работы браузера cookie-запись удаляется;
- path — строка, содержащая путь cookie-записи; если не указан, то используется путь вызванного документа;
- domain — строка, содержащая домен нужной cookie-записи; если не указан, то используется домен вызванного документа;
- secure — логическое значение (true или false), определяющее необходимость использования безопасного HTTP-соединения.

Обратите внимание, что в теле функции writeCookieQ происходит кодирование значения параметра value с помощью встроенной функции **escapeQ.**

Для удаления cookie-записи можно использовать следующую функцию:

```
function deleteCookie(name, path, domain) {
/* удаление cookie-записи */
   if (readCookie(name)) {
      document .cooki e =
      name + "=" +
      ((path) ? "; path=" + path : "") +
      ((domain) ? "; domain="+domain : "") +
      "; expires=Thu, 01-Jan-70 00:00:01 GMT"
   }
   }
```

Эта функция устанавливает дату срока хранения cookie-записи так, что запись будет удалена. Параметр path должен иметь такое же значение, которое использовалось при создании cookie-записи, или иметь пустое значение (null), если при

создании записи он не был определен. Таким же образом задается значение параметра domain.

Советую вам поэкспериментировать с чтением, созданием и удалением cookie-записей (листинг 2.4). При этом следует иметь в виду, что созданные или измененные cookie-записи будут записаны на диск только после закрытия браузера. До этого записи существуют лишь в кэше (оперативной памяти). С другой стороны, cookie-записи загружаются в оперативную память и становятся доступны как значение свойства **document.cookie** только при запуске браузера.

Листинг 2.4. HTML-код для эксперимента со сценарием, записывающим и читающим cookie-запись

```
<HTM1 >
<SCRIPT>
function readCookie(name)   {              // чтение cookie-данных записи
 . . .                                     // код

function getCookieVal(n){                  /* вспомогательная функция,
                                              вызываемая из readCookieO */
 . . .                                      // код
i

function writeCookie(name. value, expires, path, domain, secure) { /*
запись cookie */
 . . .                                     // код
}

/ * Срок хранения - 1 год от текущей даты: */
var dl = new Date ()
var d2 = d.getTimeO + (365*24*60*60*1000)
dl.setTime(d2)
/* Запись и чтение cookie: */
writeCookie( "myrecord","Привет ",dl)
alert (readCookie("myrecord") )
</SCRIPT>
</HTML>
```

Вы можете организовать для зарегистрированных посетителей вашего сайта доступ к специальным страницам, предназначенным только для них. В этом случае сценарий проверяет наличие в cookie-файле записи с некоторым фиксированным именем (например, myspecrecord)ичитаетеезначение, содержащее пароль. Если пароль правильный, то в документ загружается ссылка на защищенный паролем документ либо сам этот документ. Если такой cookie-записи не нашлось либо пароль в этой записи не верен, то в документ загружается поле ввода пароля. Если введенный пользователем пароль верен, то сценарий создает cookie-запись с именем myspecrecord и записывает в нее пароль. После этого в текущий документ загружается либо ссылка, либо сам защищенный документ. При следующем посещении этого сайта пользователю не придется снова вводить пароль, поскольку сценарий просто считает его из cookie-записи. Это, конечно, справедливо до тех пор, пока cookie-запись с паролем сохраняется на диске компьютера пользователя. Поэтому ее можно обновлять с помощью сценария, указывая новое значение срока хранения, вычисленное на основе текущей даты (например, текущая дата плюс месяц).

ВНИМАНИЕ

Значение пароля не должно фигурировать в сценарии в явном виде, а записывать пароль в cookie-файл желательно в зашифрованном виде.

Несложный алгоритм преобразования пароля вы можетепридумать самостоятельно. Другой вариант защиты страниц сайта с помощью пароля рассмотрен в главе 4, в подразделе 4.8.10.

Для закрепления полученных навыков рекомендую вам написать сценарий, использующий cookie-запись для проверки, была ли обновлена веб-страница с момента последнего ее посещения пользователем. Если вы решитесь на это, то советую еще раз обратиться к свойствам объекта Date (см. подраздел 1.7.5).

Глава 3. Объектная модель браузера и документа

В предыдущей главе вы уже познакомились с понятием и основными элементами объектной модели браузера и документа (рис. 3.1). Материал этой главы носит обзорно-справочный характер. Он понадобится вам при чтении главы 4, посвященной примерам сценариев. Заметим, что здесь рассматриваются далеко не все объекты, а лишь основные.

3.1. Объект window

Объект window содержит коллекцию frames всех фреймов, заданных в HTML-документе с помощью контейнерного тега <FRAMESET>. Объект window имеет свойства, методы, события, а также дочерние объекты. Приведем их полные перечни и рассмотрим, с разной степенью подробности, только наиболее важные с практической точки зрения.

3.1.1. Свойства window

- parent — возвращает родительское для текущего окно;
- self— возвращает ссылку на текущее окно;
- top — возвращает ссылку на главное окно;
- name — название окна;
- opener — окно, создаваемое текущим;
- closed — сообщает, если окно закрыто;
- status — текст, показываемый в строке состояния браузера;
- **defaultStatus** — текст по умолчанию строки состояния браузера;
- **returnValue** — позволяет определить возвращенное значение для события или диалогового окна;
- client — ссылка, которая возвращает объект навигатора браузеру;
- document — ссылка только для чтения на объект окнаdocu **ment;**
- event — ссылка только для чтения на глобальный объект event;
- history — ссылка только для чтения на объект окна history;

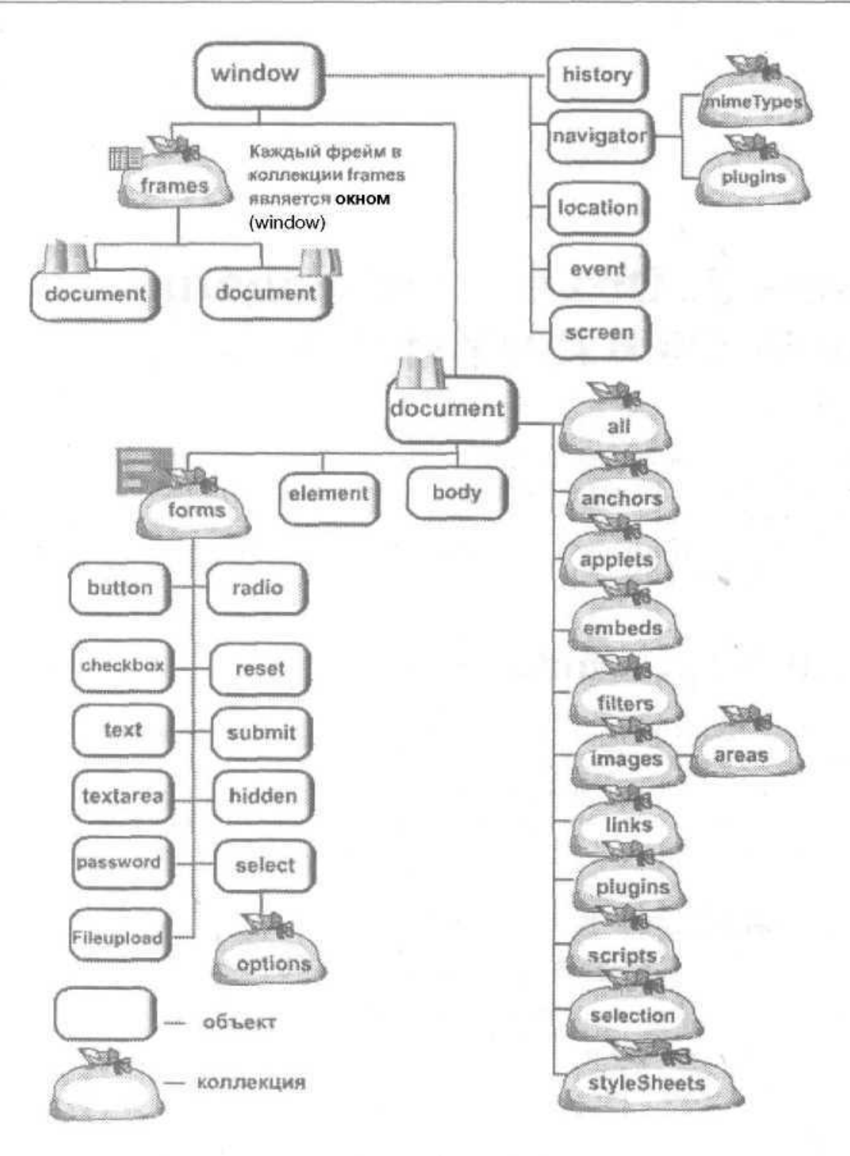


Рис. 3.1. Объектная модель

- location — ссылка только для чтения на объект окна Location;
- navigator — ссылка только для чтения на объект окна navigator;
- screen — ссылка только для чтения на глобальный объект screen.

Свойство parent позволяет обратиться к объекту, находящемуся в иерархии на одну ступень выше — например, к окну, содержащему коллекцию фреймов, в которой находится наш фрейм. Для перемещения на две ступени вверх мы должны использоватьparent.parentит.д.:

```
parent,window.frames(0)
parent.parent.window.frames(0)
```

Чтобы обратиться к самому главному окну, то есть к окну браузера, следует использовать свойство top. Однако top не может обращаться к главному фрейму вашей системы разбиения на фреймы.

Свойство name соответствует имени фрейма, которое мы задаем в теге <FRAMESET>.

Свойство status полезно использовать для вывода сообщений во время работы сценария, например при отладке:

```
window.51a1из="Сейчас работает сценарий"
```

3.1.2. Методы window

- openQ — открывает новое окно браузера;
- close() — закрывает текущее окно браузера;
- showHelpQ — показывает окно подсказки как диалоговое;
- showModalDialog() — показывает новое окно как диалоговое (модальное);
- alertQ — показывает окно предупреждения с сообщением и кнопкойOK;
- promptQ — показывает окно приглашения с сообщением, текстовым полем и кнопками OK и Cancel (Отмена);
- confirmQ— показывает окно подтверждения с сообщением и кнопками OK и Cancel (Отмена);
- navigateQ — загружает другую страницу с указанным адресом;
- blurQ — убирает фокус с текущей страницы; соответствующее событие — onblur;
- focus () — устанавливает страницу в фокус; соответствующее событие — onfocus;
- scrollQ — разворачивает окно на заданные ширину и высоту;
- setlntervalQ — указывает процедуре выполняться периодически через заданное количество миллисекунд;
- setTimeoutQ — запускает программу через заданное количество миллисекунд после загрузки страницы;
- clearlntervalQ — обнуляет таймер, заданный методомsetIntervalQ;
- clearTimeoutQ — обнуляет таймер, заданный методомsetTimeout();
- execScriptQ — выполняет код сценария; по умолчаниюJScript.

3.1.3. События window

- onblur — выход окна из фокуса;
- onfocus — окно становится активным;
- onhelp — нажатие пользователем клавиши F1;
- onresize — изменение пользователем размеров окна;
- onscroll — прокрутка окна пользователем;
- onerror — ошибка при передаче;

- onbeforeunload — передвыгрузкой страницы, чтопозволяетсохранитьданные;
- onload — страница полностью загружена;
- onunload — непосредственно перед выгрузкой страницы.

Три из перечисленных выше событий происходят в результате действий пользователя. Если открыто несколько окон браузера, пользователь может переключаться между ними, переводя фокус с одного окна на другое. Эти действия инициируют события onbluru onfocus. Заметим, что эти же события можно вызвать программным способом, используя методы blur и focus. Если происходит ошибка при загрузке страницы или ее элемента, то инициируется событие onerror. Мы можем использовать это событие в программе, чтобы, например, попытаться еще раз загрузить страницу или как-то изменить дальнейшие действия.

Пример

```
<SCRIPT>
function window.onerror(){
alert("Произошла ошибка! Попробуйте еще раз")
}
</SCRIPT>
```

Событие onload происходит, когда страница полностью загружена в окно; событие onbeforunload—nepeflTeMKaKCTpaHHnanoKHHeTOKHo;co6brraeonunload—когда страница выгружена, перед загрузкой новой страницы или перед закрытием браузера.

Пример

```
<SCRIPT>
function window.onunload() {
alert("Страница выгружается!")
}
</SCRIPT>
```

Объект window имеет несколько дочерних объектов, которые доступны с его помощью: document, history, navigator, location, event и screen.

3.2. Объект document

Объект document является центральным в иерархической объектной модели и представляет всю информацию о HTML-документе с помощью коллекций и свойств. Он также предоставляет множество методов и событий для работы с документами. Поскольку мы уже рассматривали некоторые основные приемы обращения с этим объектом, то здесь ограничимся лишь справочными сведениями.

3.2.1. Свойства document

Свойство	Атрибут	Назначение
activeElement		Идентифицирует активный элемент
alinkColor	ALINK	Цвет активных ссылок на странице
bgColor	BGCOLOR	Определяет цвет фона элемента
body		Ссылка только для чтения на неявный основной объект документа, определенный втеге<BODY>

Свойство	Атрибут	Назначение
cookie		Строка cookie-записи. Присвоение нового значения этому свойству приводит к записи cookie на диск после закрытия браузера
domain		Устанавливает или возвращает домен документа для его защиты или идентификации
fgColor	TEXT	Устанавливает цвет текста переднего плана
lastModified		Дата последнего изменения страницы, доступна как строка
linkColor	LINK	Цвет еще не посещенных гиперссылок на странице
location		Полный URL документа
parentWindow		Возвращает родительское окно для документа
readyState		Определяет текущее состояние загружаемого объекта
referer		URL страницы, которая вызвала текущую
selection		Ссылка только для чтения на дочерний для document объект selection
title	TITLE	Определяет справочную информацию для элемента, используемую при загрузке или во всплывающей подсказке
url	URL	URL-адрес документа клиента или в теге <META>
vlinkColor	VLINK	Цвет посещенных ссылок на странице

3.2.2. Коллекции document

- all — коллекция всех тегов и элементов в основной части документа;
- anchors — коллекция всех «якорей» (закладок) в документе;
- applets — коллекция всех объектов в документе, включая встроенные элементы управления, графические элементы, апплеты, внедренные и другие объекты;
- embeds — коллекция всех внедренных объектов в документе;
- forms —коллейция всех форм на странице;
- frames — коллекция всех фреймов, определенных в **теге<FRAMESET>;**
- images — коллекция всех графических элементов (изображений) на странице;
- links — коллекция всех ссылок и блоков <AREA> на странице;
- plugins — еще одно название для коллекции внедренных объектов документа;
- scripts — коллекция всех разделов <SCRIPT> на странице;
- stylesheets — коллекция всех конкретных свойств стиля, определенных в документе.

3.2.3. Методы document

- clear — очищает выделенный участок;
- close — закрывает текущее окно браузера;

- createElement — создает экземпляр элемента для выделенного тега;
- elementFromPoint — возвращает элемент с заданными координатами;
- execCommand — выполняет команду (операцию) над выделением или областью;
- open — открывает документ как поток для обработки результатов применения методов write и writein;
- queryCommandEnabled — сообщает, доступна ли данная команда;
- queryCommandIndeterm — сообщает, если данная команда имеет неопределенный статус;
- queryCommandState — возвращает текущее состояние команды;
- queryCommandSupported — сообщает, поддерживается ли данная команда;
- queryCommandText — возвращает строку, с которой работает команда;
- queryCommandValue — возвращает значение команды, определенное для документа или объекта TextRange;
- write — записывает текст и код HTML в документ, находящийся в указанном окне;
- writein — записывает текст и код HTML, заканчивающийся возвратом каретки.

3.2.4. События document

- onafterupdate — возникает при окончании передачи данных;
- onbeforeupdate — возникает перед выгрузкой страницы;
- onclick — происходит при щелчке левой кнопкой мыши;
- ondblclick — происходит при двойном щелчке левой кнопкой мыши;
- ondragstart — происходит, когда пользователь начинает перетаскивание;
- onerror — ошибка при передаче;
- onhelp — нажатие пользователем клавиши F1;
- onkeydown — возникает при нажатии клавиши;
- onkeypress — возникает при нажатии клавиши и продолжается при удержании клавиши в нажатом состоянии;
- onkeyup — возникает, когда пользователь отпускает клавишу;
- onload — возникает при полной загрузке документа;
- onmousedown — происходит при нажатии кнопки мыши;
- omousemove — происходит при перемещении указателя мыши;
- onmouseout — происходит, когда указатель мыши выходит за границы элемента;
- onmouseover — происходит, когда указатель мыши входит на элемент;
- onmouseup — происходит, когда пользователь отпускает кнопку мыши;
- onreadystatechange — возникает при изменении свойства readystate;
- onselectstart — происходит, когда пользователь в первый раз запускает выделенную часть документа.

3.3. Объект location

Объект location содержит информацию об URL-адресе (и его компонентах) текущей страницы, а также методы, позволяющие обновлять страницы.

3.3.1. Свойства location

- href— полный URL-адрес в виде строки;
- hash — строка, следующая в URL за символом#, с помощью которого указывается, к какому анкеру следует переместиться при загрузке документа.;
- host — часть URL«хосг.порт»; значение порта содержится лишь тогда, когда оно явно было указано в URL-адресе;
- hostname — часть URL «хост»;
- pathname — путь к объекту или файлу, находящийся после третьего«/»;
- port — номер порта URL;
- protocol — начальная часть, определяющая протокол, за которой следует двоеточие, например «http:»;
- search — строка запроса или данные URL после знака«?».

Например, если вы загрузили страницу с адресом http://www.piter.com, то значением location.href будет эта строка.

Присваивая свойству href новое значение, мы можем изменять показываемую в браузере страницу, например:

```
window.location.href = "http://www.rambler.ru"
```

3.3.2. Методы location

- assign() — загружает другую страницу; этот метод эквивалентен изменению свойства window.location.href;
- reloadQ — обновляет текущую страницу;
- replaceQ — загружает страницу с указанным в параметре URL-адресом и заменяет URL-адрес текущей страницы (location.href).

Некоторые веб-сайты могут содержать страницы, которые не предназначены для посещения пользователем и занесения их в список посещенных страниц. Например, перемещение по сайту может привести пользователя на некоторые промежуточные страницы, которые ему больше не понадобятся. При этом желательно, чтобы пользователь не мог вернуться к ним снова с помощью кнопки Назад (Back). В этом случае используют метод location.replace(URL-aflpec) для перехода к указанной странице без занесения его в список документов, переход к которым осуществляется щелчком на этой кнопке.

3.4. Объект history

Объект history содержит информацию об адресах страниц, которые браузер посетил во время текущего сеанса. Мы можем передвигаться по этому списку с по-

мощью сценария и загружать соответствующие страницы. Объект history имеет только одно свойство и три метода.

3.4.1. Свойство history

length — количество элементов в списке посещенных страниц.

Это свойство используется в сценарии для контроля того, чтобы не выйти за пределы списка.

3.4.2. Методы history

- back() — загружает предыдущую страницу из списка;
- **forwardQ** — загружает следующую страницу из списка;
- до() — загружает страницу с относительным номером п из списка (от 0 до history.length-1) или с указанным URL-адресом.

В следующем примере загружается первая страница, а затем загружается пятая страница, причем предварительно проверяется, существует ли она в списке уже посещенных страниц:

```
window.hi story.go(l)
if  (window.history.1ength>4)
      window.hi story.go(5)
```

Очевидно, метод history.go(-l) эквивалентен history.backQ, а метод window.history. go(l) эквивалентен методу history.forwardQ.

3.5. Объект navigator

Объект navigator содержит информацию о производителе браузера, его версии и возможностях.

3.5.1. Свойства navigator

- appCodeName — название кода браузера; например"Mozilla";
- appName — название браузера; например "Microsoft Internet Explorer";
- appVersion — версия браузера; например "4.0 (compatible;**MSIE** 6.0; Windows 98; Win **9x4.90)";**
- cookieEnabled — определяет возможность использования в браузере cookies со стороны клиента; логическое значение;
- userAgent — название браузера, посылаемое с **помощьюhttp-протокола;** например **"Mozilla/4.0** (compatible; MSIE 6.0; Windows 98; Win 9x 4.90)".

3.5.2. Коллекции navigator

- **mimeTypes** — коллекция всех типов документов и файлов, поддерживаемых браузером;
- **plugins** — коллекция всех внедряемых объектов на странице.

3.5.3. Методы navigator

- taintEnabled — возвращает значение false, включен для совместимости с Netscape Navigator;
- javaEnabled — сообщает, возможен ли в данном браузере запуск кода сценария на языке JavaScript; логическое значение.

3.6. Объект event

Объект event позволяет получить информацию о каком-либо событии, происходящем в браузере. Эта информация содержится в следующих свойствах:

- altKey — возвращает состояние клавиши Alt, когда происходит событие;
- button — кнопка мыши, вызывающая событие;
- cancelBubble — устанавливается для запрета прохождения заданного события вверх по объектной иерархии;
- clientX — возвращает координату x элемента, исключая обрамление, отступы, полосы прокрутки и т. д.;
- clientY — возвращает координату y элемента, исключая обрамление, отступы, полосы прокрутки и т. д.;
- CtrlKey — состояние клавиши Ctrl при появлении события;
- from Element — возвращает элемент, с которого ушел курсор мыши, для событий onmouseover и onmouseout;
- keyCode — код ASCII нажатой клавиши; есть возможность изменять значение, передаваемое объекту;
- offsetX — возвращает координату x указателя мыши в пикселах относительно содержащего его элемента при возникновении события;
- offsetY — возвращает координату y указателя мыши в пикселах относительно содержащего его элемента при возникновении события;
- reason — указывает, что перемещение данных прошло успешно или из-за чего онопрекратилось;
- returnValue — определяет возвращаемое значение для события;
- screenX — возвращает горизонтальную координату указателя мыши относительно экрана, когда происходит событие;
- screenY — возвращает вертикальную координату указателя мыши относительно экрана, когда происходит событие;
- shiftKey — определяет состояние клавиши Shift при возникновении события;
- srcElement — возвращает элемент, с которого началось прохождение события;
- srcFilter — возвращает фильтр, создавший событиеonfilterchange;
- toElement — возвращает элемент, на который наезжает курсор мыши, при появлении события onmouseover или onmouseout;
- type — возвращает название события как строку, без приставки on;

- x — возвращает координату x указателя мыши соответственно либо к позиционированному родительскому элементу, либо к окну;
- y — возвращает координату y указателя мыши соответственно либо к позиционированному родительскому элементу, либо к окну.

Свойства объекта event устанавливаются в момент прохождения события и большинство из них доступны только для чтения (их нельзя изменить). Однако есть два изменяемых свойства: keyCode и **return** Value. Мы уже рассматривали применение объекта event.

3.7. Объект screen

Объект screen содержит информацию о возможностях экрана пользователя и может применяться, например, для определения размеров создаваемых окон и выбора разрешения передаваемой графики (нет смысла загружать 32-битное изображение, если монитор и видеокарта пользователя поддерживают только 256 цветов). Объект screen имеет следующие свойства:

- width — возвращает ширину экрана пользователя (в пикселах);
- height — возвращает высоту экрана пользователя (в пикселах);
- **bufferDepth** — определяет, используется ли буфер для хранения «второго экрана»;
- colorDepth — возвращает информацию, позволяющую решить, как использовать цвета на экране; количество бит на пиксел устройства или видеобуфера пользователя;
- **updatelnterval** — возвращает или задает интервал времени между обновлениями экрана пользователя.

3.7.1. Объект TextRange

Объект TextRange (текстовая область) отображает разделы потока текста, формирующего HTML-документ. Может использоваться для управления текстом внутри страницы.

3.7.2. Свойства TextRange

- htmlText — возвращает содержимое TextRange как текст и код HTML;
- text — простой текст, находящийся внутри элемента TextRange илитега<ОРПОМ>.

3.7.3. Методы TextRange

- collapse —стягивает текстовую область в точку в начале или конце текущей области;
- **compareEndPoints** — сравнивает две текстовые области и возвращает значение, показывающее результат;

- duplicate — возвращает копию области TextRange;
- execCommand — выполняет команду (операцию) над выделением или областью;
- expand — расширяет текстовую область, добавляя туда новый знак, слово, предложение, или указывает, какие неполные блоки полностью в ней содержатся;
- findText — определяет текстовую область, содержащую только искомый текст;
- getBookmark — возвращает значение, позволяющее в дальнейшем идентифицировать данную позицию в документе;
- inRange — определяет, находится ли заданная текстовая область внутри текущей;
- isEqual — определяет, равны ли заданная и текущая текстовые области;
- move — изменяет начальную и конечную точки текстовой области для включения в нее различного текста;
- moveEnd — заставляет текстовую область сжаться или расшириться до заданной конечной точки;
- moveStart — заставляет текстовую область сжаться или расшириться до заданной начальной точки;
- moveToBookmark — передвигает границы текстовой области для включения другой, определенной ранее с помощью getBookmark;
- moveToElementText — передвигает границы текстовой области для включения текста в заданном элементе;
- moveToPoint — передвигает границы текстовой области и сжимает ее вокруг выбранной точки;
- parentElement — возвращает элемент, родительский по отношению ко всему, что входит в текстовую область;
- pasteHTML —вставляет текст и/или HTML-код в текущую текстовую область;
- queryCommandEnabled — сообщает, доступна ли данная команда;
- queryCommandlndeterm — сообщает, если данная команда имеет неопределенный статус;
- queryCommandState — возвращает текущее состояние команды;
- queryCommandSupported — сообщает, поддерживается ли данная команда;
- queryCommandText — возвращаетстроку, с которой работает команда;
- queryCommandValue — возвращает значение команды, определенное для документа или объектаText Range;
- scrollIntoView — переносит текущую текстовую область в видимую часть окна браузера;
- select — делает активный подсвеченный участок выделения на странице равным текущей текстовой области;
- setEndPoint— переносит начальную или конечную точку текущей текстовой области в начало или конец заданной области.

Глава 4. Примеры сценариев

В этой главе мы рассмотрим примеры решения некоторых задач. Не все из них одинаково часто возникают при разработке приложений и, в частности, веб-страниц, однако их изучение позволит получить практические навыки программирования. Кроме того, некоторые примеры могут вдохновить вас на разработку собственных проектов.

4.1. Простые визуальные эффекты

4.1.1. Смена изображений

В веб-дизайне часто возникает необходимость заменить одно изображение на другое. Новички нередко начинают свое творчество именно с этой операции. Поэтому рассмотрим данную задачу подробнее.

Суть смены изображений заключается в том, чтобы с помощью сценария изменить значение атрибута SRC тега <IMG>. Напомним, что атрибут SRC имеет в качестве значения строку, указывающую месторасположение графического файла. Если элемент<IMG>, задающий изображение, содержится в HTML-документе, тов объектной модели имеется объект этого элемента со свойством src. Значение этого свойства можно изменить в сценарии. При этом в окно браузера загружается соответствующий графический файл, если, разумеется, он будет найден. В следующем примере щелчок на изображении из файла pictl.gif заменяет его изображением из файла pict2.gif. Поскольку сценарий очень небольшой, он записан в строке, которая присваивается атрибуту onclick тега <IMG>.

```
<HTML>
<IMG ID = "myimg" SRC = 'pictl.gif
onclick = "document.all.myimg.src = 'pict2.gif'">
</HTML>
```

В приведенном выше примере смена изображения происходит лишь при первом щелчке на нем. Последующие щелчки не приведут к видимым изменениям, поскольку второе изображение будет заменяться им же. Чтобы повторный щелчок приводил к отображению предыдущего рисунка, сценарий необходимо слегка усложнить: следует создать переменную-триггер (так называемый флаг), принимающий одно из двух возможных значений. По текущему значению флага сценарий может определить, какое именно из двух изображений следует отобразить. После смены изображения необходимо изменить и значение флага. Далее приведен вариант кода:

```
<HTML>
<IHG ID = "myimg" SRC = 'pictl.gif onclick = "imgchangeQ ">
<SCRIPT>
var flag=false             // флаг (триггер)

function imgchange() {     // обработчик щелчка на изображении
if (flag) document.all.myimg.src = "pictl.gif"
  else document.all.myimg.src = "pict2.gif"
flag=!flag                 // изменяем значение флага на противоположное
}
</5CRIPT>
</HTML>
```

Думаю, с изменением одного изображения все ясно. А как быть в случае нескольких изображений? Например, на веб-странице расположена галерея миниатюр (thumbnails — уменьшенных копий крупномасштабных изображений). Мы хотим, чтобы при щелчке кнопкой мыши на миниатюре она увеличивалась, а затем при щелчке на увеличенном изображении оно уменьшалось. Для решения этой задачи потребуется столько флагов, сколько имеется изображений. Флаги определим как массив, каждый элемент которого будет соответствовать отдельному изображению. Далее мы не будем создавать отдельную функцию-обработчик события onclick для каждого графического объекта. Вместо этого создадим одну функцию-обработчик, которая кроме всего прочего будет сама определять, на каком именно изображении произошел щелчок. Идентификаторы ID тегов <IMG> зададим некоторым регулярным образом (например "10", "il", "i2", ...). Так часто поступают при использовании массивов. Создадим еще два массива, содержащих имена графических файлов: один для исходных изображений, а другой — для замещающих. Наконец, HTML-документ с изображениями сгенерируем с помощью сценария. Для этого сначала сформируем строку, содержащую теги <IMG ...>, а затем запишем ее в документ. В листинге 4.1 приводится код, реализующий эту программу.

Листинг 4.1. Код программы для увеличения миниатюр при щелчке мыши

```
<HTML>
<SCRIPT>
var apictl = new Array("pictl.gif"   ,...)   /* массив имен исходных
                                                 файлов */
var apict2 = new Array("pict2.gif"   ,...)   /* массивимензамещающих
                                                 файлов */

var aflag = new Array(apictl.length)       // массив флагов
/* Формирование строки тегов, описывающих изображения */
var xstr = ""
for(i =0; i < apictl.length; i++){
xstr+= '<IMG ID = "i' + i + '" SRC = "' + apictl[i]+'" onclick =
"imgchangeO">'
}
document.write(xstr)             // запись в документ

function imgchangeO {            // обработчик щелчка на изображении
var xid = event.srcElement.id    // id изображения, на котором был щелчок
var n = parselnt(xid.substr(1)) // выделяем номер элемента

if (aflagtn])
document.all[xid].src = apictl[n]
```

продолжение #

Листинг 4.1 *(продолжение)*

```
else
document.all[xid].src = apict2[n]
aflag[n] = !aflag[n]                    /* изменяем значение флага
                                           на противоположное */
|
</SCRIPT>
</HTML>
```

Обратите внимание, как мы обращаемся к свойству src: document.allfxid].src, а не document.all.xid.src и тем более не document.all["xid"].src. Это объясняется тем, что xid является строковой переменной, содержащей значение идентификатора ID, а не собственно значением идентификатора.

4.1.2. Подсветка кнопок и текста

Рассмотрим задачу изменения цвета кнопки при наведении на нее указателя мыши. При удалении указателя с кнопки должен вернуться ее первоначальный цвет. Это так называемая подсветка кнопок. Сразу займемся общим случаем нескольких кнопок. В нашем примере три элемента, задающие кнопки, находятся в контейнере формы <FORM>. К этому контейнеру привязываются обработчики событий onmouseover (наведение указателя мыши) и onmouseout (удаление указателя мыши). Таким образом, инициатором (получателем) этих событий может быть любой элемент формы (в нашем примере — любая из трех кнопок). В обычном состоянии кнопки имеют серый цвет, заданный 16-м кодом aOaOaO. При наведении указателя мыши цвет кнопки становится желтым (yellow) (рис. 4.1).

```
<HTML>
<STYLE>
mystyle {font-weight: bold ; background-color : aOaOaQ}
</STYLE>
                                                    #
<FORM onmouseover = "colorchange('yellow')" onmouseout
  ="colorchange('aOaOaO')">
<INPUT TYPE = "BUTTON '" VALUE="nepBaa" CLASS = "mystyle" onclick =
  "alert('Bbi нажали кнопку')">
<INPUT TYPE = "BUTTON" VALUE = "Вторая" CLASS = "mystyle" onclick =
  "alert('Вы нажали кнопку 2')">
<INPUT TYPE = "BUTTON" VALUE = "Третья" CLASS = "mystyle" onclick =
  "alert('Bbi нажали кнопку 3')">
</FORM>
<SCRIPT>
function colorchange(color){                  // изменение цвета кнопок
if (event.srcElement.type == "button")
   event. srcElement.style.backgroundColor = color;
|
</SCRIPT>
</HTML>
```

Здесь в функции colorchangeQ проверяется, является ли инициатор события объектом типа button (кнопка). Если это так, то цвет кнопки изменяется, в противном случае — нет. Без этой проверки изменялся бы цвет не только кнопок, но и фона.

Аналогичным образом можно изменять цвет и других элементов, например фрагментов текста. Если требуется подсвечивать текст, то он должен быть заключен в какой-нибудь контейнер, например в теги <P>, <B>,<1> или <DIV>. В следующем

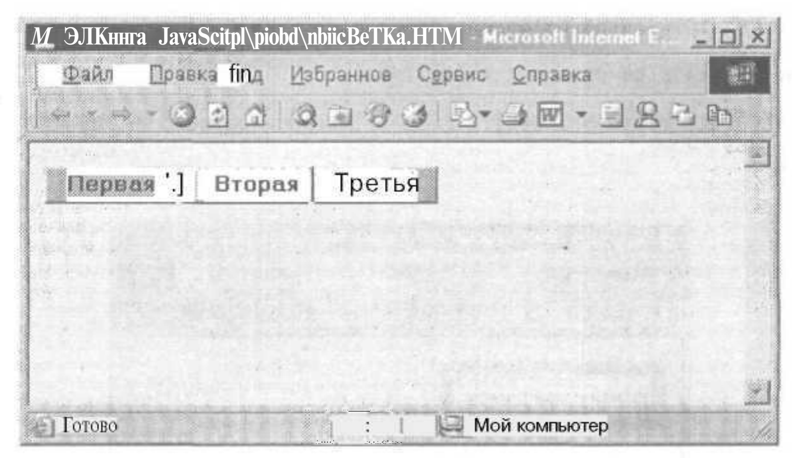


Рис. 4.1. Подсветка кнопки при наведении на нее указателя мыши

примере цвет текста, заключенного в тег <B>, изменяется с синего на красный при наведении на него указателя мыши:

```
<HTML>
Этот <B onmouseove r= "colorch('red ')" onmouseout = "colorch('blue ')"
style="color: Ыие">Жирный текст</B> при наведении на него указателя мыши
изменяет цвет
<SCRIPT>
function colorch(color){
event.srcElement.style.color = color
}
</SCRIPT>
</HTML>
```

4.1.3. Мигающая рамка

Вы можете создать прямоугольную рамку, окаймляющую некий текст, которая периодически изменяет цвет. Иногда этот эффект используют для привлечения внимания пользователей. Рамка создается тегами одноячеечной таблицы с заданием нужных атрибутов и параметров стиля. Далее необходимо создать функцию, изменяющую цвет рамки таблицы на другой, и передать ее в качестве первого параметра методу setIntervalQ. Второй параметр этого метода задает период в миллисекундах, с которым вызывается функция, указанная в первом параметре. Более подробно этот метод описан в разделе 2.8.

В приведенном ниже примере рамка изменяет цвет с желтого на красный с периодом 0,5 с (рис. 4.2):

```
<HTML>
<TABLE ID="mytab  BORDER=1 WIDTH=150 style="border: 10 solid:yellow">
<TR><TD>Мигающая раМКа</TD></TR>
</TABLE>
<SCRIPT>
function flash() {                  // изменение цвета рамки,
if ((document.all)                  // если в документе ничего нет
return null;
if (mytab. style.borderColor == 'yellow')
```

```
    raytab.style.borderColor= 'red'
else
    mytab. style.borderColor = 'yellow';
}
setlnterval("flash() " , 500);        // мигание рамки с интервалом 500 мс
</SCRIPT>
</HTML>
```

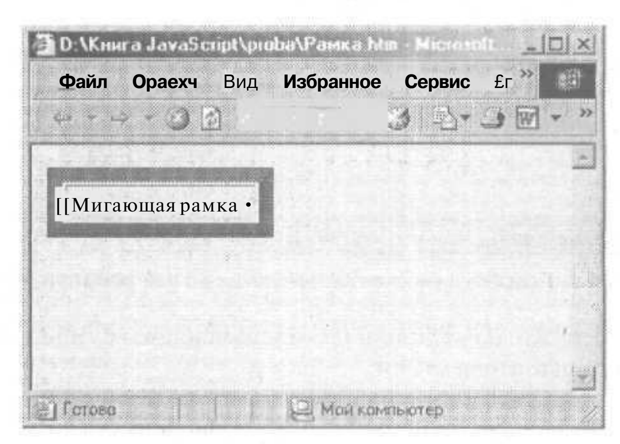


Рис. 4.2. Мигающая рамка

4.1.4. Переливающиеся цветами ссылки

Привлечь внимание посетителей к ссылкам на веб-странице можно с помощью эффекта динамического изменения их цвета. Мы не будем обсуждать здесь целесообразность этого метода, а сосредоточимся лишь на технической стороне. Суть задачи состоит в том, чтобы случайным образом выбирать и устанавливать цвет ссылок. В листинге 4.2 множества цветов, из которых происходит выбор, различаются для уже использованных и еще не использованных ссылок. Эти множества цветов задаются в виде массивов.

Листинг 4.2. Код для динамического изменения цвета ссылок

```
<HTML>
<A HREF="http://www. admi ral. ru/~dunaev/i_is . Шт">Начало</A>
<A HREF=" http://www. admi ral. ru/~dunaev/examples . 1пт_т">Примеры HTML и
JavaScript</A>
<A HREF=" http://www.admiral.ru/~dunaev/mybook.htm">МоН книги</A>
<SCRIPT>
aclrlink=new ArrayO              // массив цветов неиспользованных
                                 // ссылок
aclrlink [0] = 'yellow'
aclrlink [1] = '#80FF80'
aclrlink [2]= '#FFFF80 '
aclrlink [3]='#408000'
aclrvlink k=new ArrayO           // массив цветов использованных ссылок
aclrvlink [0]='blue'
aclrvlink [1) = 'purple'
aclrvlink [2] = ' black'
aclrvlink [3] ='red'
```

```
function colorchangeO {           // изменение цвета
link=Math.round((aclrlink.length+0.l)*Math.randomO))
vlink= Math, round ((aclrvlink. length+0. l)*Math.randomO)

    document.linkColor= aclrlink [link]
    document. vli nkColor= aclrvlink [vlink]
}
setlnterval ("colorchange() ", 500)  // изменение цвета через 500 мс
</SCRIPT>
</HTML>
```

Заметим, что в результате вставки этого сценария в HTML-документ все ссылки документа начинают переливаться разными цветами, поскольку мы изменяем свойства HnkColor и vlinkColor объекта document.

4.1.5. Объемныезаголовки

Объемные (выпуклые) заголовки довольно эффектны на веб-страницах. Зачастую они создаются как графические файлы, которые вставляются в HTML-документ с помощью тега <IMG>. Однако во многих случаях более экономичным является решение, основанное на использовании таблиц стилей.

Идея создания объемного заголовка довольно проста: достаточно несколько надписей с одинаковыми содержанием наложить друг на друга с некоторым сдвигом по координатам. Потребуются как минимум две такие надписи. Одна из них предназначена для создания эффекта тени (задний план), а вторая располагается над первой. Можно использовать еще одну такую надпись для создания эффекта подсветки. С эстетической точки зрения наилучший эффект достигается путем подбора цветов надписей (игрой света и тени) с учетом цвета фона. С технической точки зрения нужный эффект получается применением таблиц стилей. В листинге 4.3 приведен пример HTML-документа, в котором объемный заголовок создается с помощью трех наложенных друг на друга надписей (рис. 4.3). Таблица стилей определяется для тега абзаца <P>. В данной таблице задаются цвет и параметры шрифта надписей. Позиционирование надписей производится параметрами атрибута STYLE контейнерных тегов <DIV>, в которые заключены теги абзаца.

Листинг 4.3. Код для создания объемного заголовка

```
<HTML>
<HEAD><TITLE>3d эффект</TITLE><HEAD>
<! Каскадная таблица стилей для абзаца>
<STYLE>
    P {font-family:sans-serif; font-size:72; font-weight:800;
color:00aaaa}
    P.highlight {color:silver}
    P.shadow {coloridarkred}
</STYLE>
<BODY BGCOLOR = aeb98c>
<! Тень >
<DIV STYLE = "posi tion : absolute ; top:5; left:5">
<P CLASS = shadow>Объемный заголовок</P>
</DIV>
```

*продолжение *у*

Листинг 4.3 *(продолжение)*

```
<! Подсветка >
<DIV STYLE = "position:absolute; top:0; left:0">
<P CLASS = highlight>Объемный заголовок</P>
</DIV>
<! Передний план>
<DIV STYLE = "position:absolute; top:2; left:2">
<P>Объемный заголовок</P>
</DIV>
</BODY>
</HTML>
```

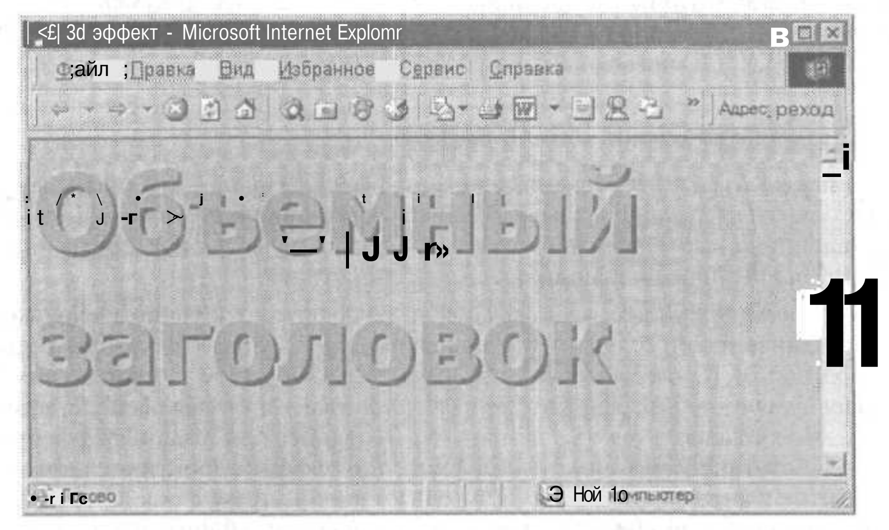


Рис. 4.3. Объемный заголовок, созданный с помощью каскадной таблицы стилей

Теперь рассмотрим функцию, которая создает заголовок с заданными параметрами (листинг 4.4).

Листинг 4.4. Пример функции, которая создает заголовок с заданными параметрами

```
function d3(text, x, y, tcolor, fsize, fweight, ffamily, zind) {
/* Объемный заголовок
   Параметры:
   text - текст заголовка
   x - горизонтальная координата (left)
   y - вертикальная координата (top)
   tcolor - цвет переднего плана
   fsize - размер шрифта (пт)
   fweight - вес (толщина шрифта)
   family - название семейства шрифтов
   zind - z-index */
if (!text) return null /* если не указан текст, то ничего не делаем */
/* Значения параметров по умолчанию: */
if (!ffamily) ffamily='arial'
if (!fweight) fweight=800
if (!fsize) fsize=36
if (!tcolor) tcolor='00aaff'
if (!y) y=0
```

```
if (!x) x=0
/* Внутренние настройки */
var sd=5 , hd=2                              // сдвиг тени и подсветки
var xzind=""
if (zind) xzind=" ;z-Index: "+zind
var xstyle=' font-family: ' + family + ';font-size: ' + fsize +
   ';font-weight: ' + fweight+';'
var xstr = '<DIVSTYLE = "position: absolute; top:' + (y + sd) +
   ' ; lef t: ' + (x + sd) + xzind + '">'
xstr+= '<P styl e = "' + xstyle + 'color:darkred">' + text + '</P></DIV>'
xstr+= '<DIV STYLE = "position: absolute; top:' + y + ';left:' + x +
  xzind + ' ">'
xstr+= '<P style ="' + xstyle + 'color:silver">' + text + '</P></DIV>'
xstr+= '<DIV STYLE = "position: absolute; top:' + (y + hd) + ' ; left:' +
   (x + hd) + xzind + '">'
xstr+= '<P style = "' + xstyle + 'color:' + tcolor + '">' + text + '
  </P></DIV>'
document.write(xstr)                         // запись в документ
}
```

В этой функции создается строка, содержащая необходимые теги, которая затем записывается в HTML-документ. Среди параметров функции последний представляет **z-Index,** с помощью которого можно установить слой, в котором находится заголовок, и тем самым указать, будет ли заголовок располагаться над или под каким-либо другим видимым элементом документа. Элементы с более высоким значением z-Index находятся над элементами, у которых z-Index меньше. Перекрытие элементов с одинаковыми значениями z-Index определяется порядком следования их тегов в HTML-документе. Значения параметров по умолчанию (то есть когда они не указаны) вы можете задать самостоятельно. Можете также поэкспериментировать со значениями внутренних параметров sd и hd, которые определяют размеры тени и ореола подсветки.

Ниже приведен пример HTML-документа со сценарием, создающим три заголовка с различными параметрами (рис. 4.4):

```
<HTML>
<HEAD><TITLE>3d эффект</TITLE><HEAD>
<SCRIPT>
function d3 (text, x, y, tcolor, fsize, fweight, ffamily, zind) {
      // код
}
d3("Привет!", 50,15,'red' ,40,800,'sans-serif')
d3("3То не графика",50,50, 'blue' ,72,800, ' times ')
d3("Это просто стиль текста" , 10, 80, '00ff00' , 92,900, 'arial' ,-7)
</SCRIPT>
</HTML>
```

4.1.6. Применение фильтров

С помощью так называемых фильтров каскадных таблиц стилей можно получить разнообразные интересные визуальные эффекты. Например, постепенное появление (исчезновение) рисунка, плавное преобразование'одного изображения в другое, задание степени прозрачности и т. п. В большинстве случаев веб-дизайнеры добиваются подобных эффектов с помощью обработки изображений сред-

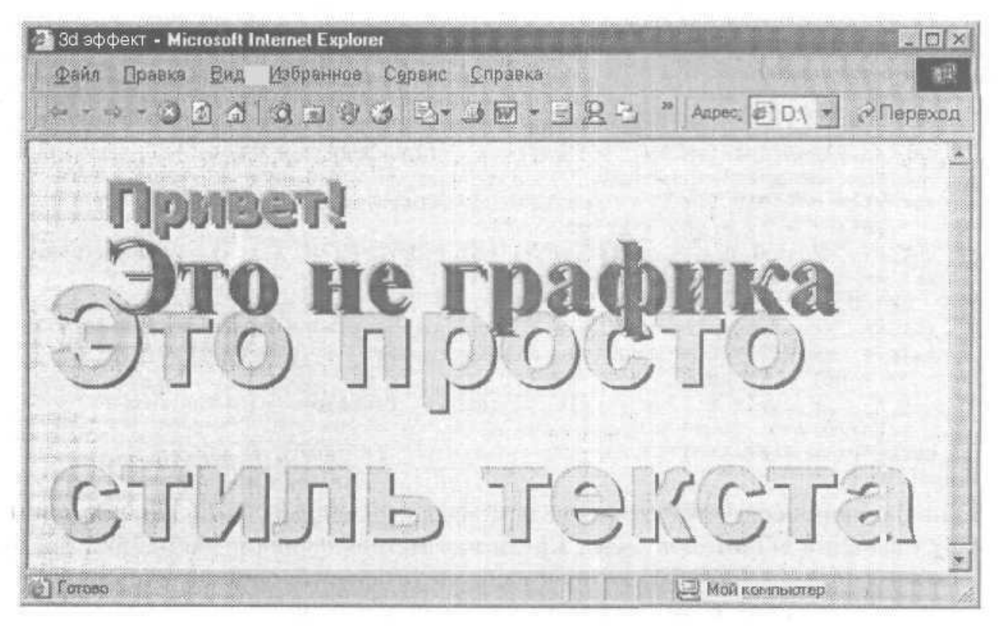


Рис. 4.4. Заголовки, созданные с помощью функции d3()

ствами графических редакторов, таких как Adobe Photoshop, Macromedia Flash и др. При этом графика для Веб обычно сохраняется в файлах формата gif, png, jpeg и swf.

- GIF очень популярен в веб-дизайне. Поддерживает чересстрочную загрузку, прозрачность пикселов и анимацию. Однако глубина цвета в GIF-файлах ограничена лишь 256 цветами, а пикселы могут быть либо полностью прозрачными, либо полностью непрозрачными (полупрозрачности быть не может).
- PNG имеет много общего с форматом GIF, но позволяет сохранять полноцветные изображения и не поддерживает анимацию.
- JPEG — еще один графический формат, часто используемый в веб-дизайне. Позволяет сохранять полноцветные изображения фотографического качества и загружать их в чересстрочном режиме, но не поддерживает прозрачность и анимацию.
- В формате SWF сохраняется векторная графика и анимация, созданные в пакете Macromedia Flash, а также импортированные растровые изображения и звуковое сопровождение. Прозрачность изображений здесь может принимать множество значений от 0 до 100.

СОВЕТ

Во многих случаях при создании небольших анимаций (например, баннеров) и других не слишком сложных визуальных эффектов можно вполне обойтись средствами таблиц стилей и JavaScript, достигая своих целей при существенно меньшихзатратах ресурсов (объем файлов, время на разработку). При разработке более сложных проектов фильтры можно комбинировать с другими средствами.

Фильтр следует понимать как некий инструмент преобразования изображения, взятого из графического файла и вставленного в HTML-документ с помощью тега <IMG>. Однако следует иметь в виду, что фильтры работают только в IE4+. В этом подразделе мы рассмотрим несколько наиболее интересных фильтров. Наша задача заключается в том, чтобы научиться правильно применять фильтры.

Прозрачность

Прозрачность графического объекта в диапазоне целочисленных значений от 0 до 100 можно установить с помощью фильтра alpha. Значение 0 соответствует полной прозрачности изображения, то есть оно становится невидимым. Значение 100 соответствует, наоборот, полной непрозрачности изображения. Поэтому правильнее говорить, что alpha является степенью непрозрачности. Сквозь прозрачные графические объекты видны нижележащие изображения. Степень их видимости определяется значением прозрачности изображения, лежащего выше. Кроме того, прозрачность имеет несколько вариантов градиентной формы. Например, можно сделать так, чтобы прозрачность постепенно увеличивалась от центра к краям изображения. Это позволяет просто и весьма эффективно согласовать вставляемое в документ изображение с фоном. Веб-дизайнеры хорошо знают, что выполнение такой операции с помощью графического редактора требует определенных навыков и усилий.

Фильтр alpha, как и другие, задается с помощью каскадной таблицы стилей и имеет ряд параметров. В следующем примере для графического изображения стиль определяется с помощью атрибута STYLE:

```
<IMG ID = "myimg" SRC = "pict.gif"
STYLE = "position:absolute; top: 10; left:50;
filter:alpha(opacity = 70. style = 3)">
```

Здесь параметр opacity определяет степень прозрачности (точнее, непрозрачности) как целое число в диапазоне от 0 до 100. Параметр style задает градиентную форму распределения прозрачности по изображению как целое число от 0 до 3. Если параметр style не указан или имеет значение 0, то градиент не применяется. Фильтр alpha имеет и другие параметры, определяющие прямоугольную область изображения, к которой применяется фильтр. По умолчанию он применяется ко всему изображению. На рис. 4.5 представлено исходное изображение и это же изображение, обработанное фильтром alpha с параметрами opacity = 70 и style = O, 1, 2, 3 (слева направо и сверху вниз).

Фильтр можно определить в каскадной таблице стилей внутри контейнерного тега <STYLE> с помощью ссылки, имеющей следующую структуру:

```
#1c)_изображения {fi lter: имя_фильтра (параметры)}
```

Заметим, что если не использовать символ #, фильтр не будет работать. В этом случае рассмотренный выше пример можно оформить следующим образом:

```
<HTML>
<STYLE>
  #myimg { position:absolute; top: 10; left: 50; filter: alpna(opacity =
    70, style = 3)}
</STYLE>
<IMG ID = "myimg" SRC = "pict.gif"
</HTML>
```

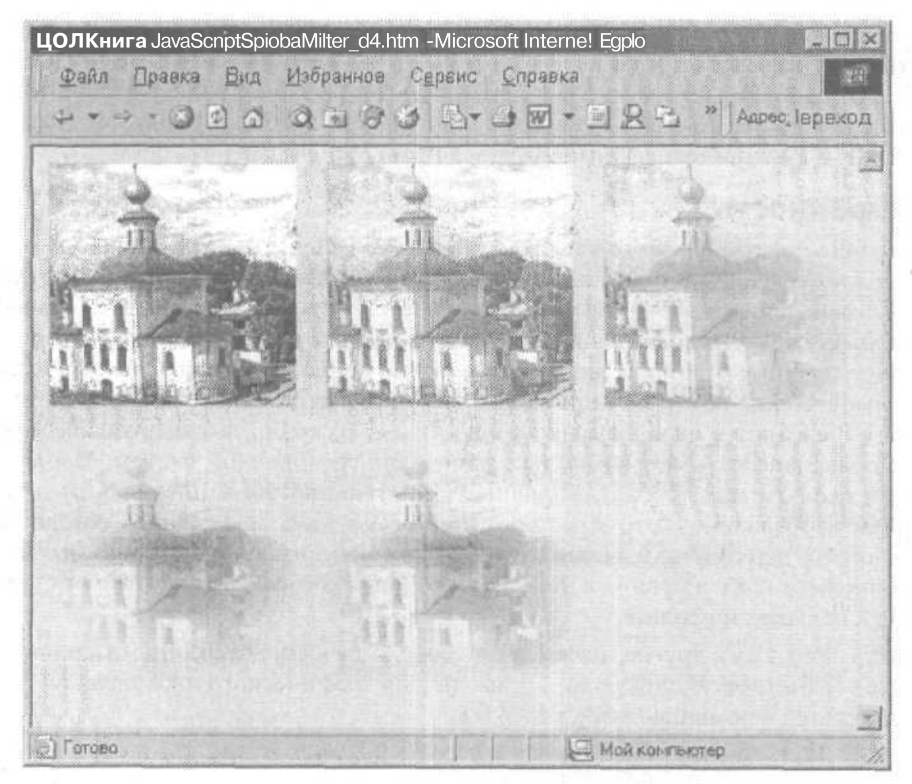


Рис. 4.5. Результаты использования фильтра alpha с различными параметрами

Доступ к свойствам фильтра в сценарии определяется следующим образом:

```
document.all.1c1_изображения.filters["имя_фильтра"].параметр = значение
```

Для рассматриваемого случая это выражение может иметь, например, такой вид:

```
document.all.myimg.filters["alpha"].opacity=30
```

Итак, вы можете управлять в сценарии степенью прозрачности и другими параметрами фильтраalpha.

Выше мырассмотрелисинтаксисвыражений, воспринимаемый браузерами IE4+. Для IE5.5+ можно использовать другой синтаксис, рекомендованный Microsoft. Его особенность в том, что в каскадной таблице стилей задается ссылка на специальный компонент и имя фильтра:

```
# 1c1_изображения {filter: progid:DXImageTransform.Microsoft.
имя_фильтра(параметры)}
```

Пример

```
# myimg {filter: progid:DXImageTransform.Microsoft.alpha
(opacity = 70, style = 3)}
```

Доступ к свойствам фильтра в сценарии в новом синтаксисе определяется следующим образом:

```
document.all.1й_изображения.filters["DXImageTransform.Microsoft.
имя_фильтра"].параметр = значение
```

Пример

```
document.all.myimg.filters["DXImageTransform.Microsoft.alpha"].opacity = 30
```

СОВЕТ

Фильтры можно применять не только к графическим изображениям, но и к другим элементам (например, к текстам, текстовым областям, кнопкам).

Трансформация

Фильтры преобразования изображений позволяют организовать постепенное появление (исчезновение) изображения, а также трансформацию одного графического объекта в другой. Это так называемые динамические фильтры. В отличие от статических фильтров (например, alpha), их применение обязательно связано со сценариями. Суть преобразования графического объекта заключается в том, что сначала необходимо зафиксировать первое изображение, затем выполнить замену этого изображения другим и/или изменить параметры того же самого изображения, а после этого выполнить собственно трансформацию. Все эти действия выполняются в сценарии. Фиксация и трансформация изображения производятся с помощью специальных методов фильтра — соответственно applyQ и play(). При желании остановить процесс преобразования можно с помощью метода **stopQ.**

Для преобразования изображений используется фильтр revealtrans. Он имеет следующие параметры:

- duration — длительность преобразования в секундах (число с плавающей точкой);
- **transition** — тип преобразования (целое число от 0 до 23); типу 23 соответствует один из типов от 0 до 22, выбранный случайным образом.

Сначала рассмотрим применение фильтра revealtrans для создания эффекта появления изображения. Ниже приводится пример, в котором изображение постепенно появляется после загрузки документа, то есть по событию onload:

```
<HTML>
<BODY onload = "transform()">
<IMG ID = "myimg"  SRC = "pict.gif" STYLE = "position:absolute; top:10;
left:50; visibility = "hidden" filter:revealtrans(duration= 3 ,
transition = 12)">
</BODY>
<SCRIPT>
function transforra0{                          // появление изображения
document.all.myimg.style.visibility = "hidden"  /* делаем изображение
                                                   невидимым */
myimg. filters C 'revealtrans") . apply0      /* фиксируем исходное
                                                 состояние изображения */
myimg.style.visibility = "visible"           // делаем изображение видимым
myimg. f iltersC ' revealtrans") .play()      // выполняем преобразование
}
</SCRIPT>
</HTML>
```

В этом примере, в функции transform(), мыделаем изображение, заданное в HTML-документе тегом **<IMG>,** сначала невидимым, поскольку хотим, чтобы оно возникало из небытия, а не исчезало. Затем с помощью метода apply() мы фиксируем

его как исходный кадр в цепочке кадров преобразования. Далее готовим конечный кадр — просто делаем изображение видимым. Наконец, применяем фильтр с помощью метода play(). Мы использовали тип преобразования (transition = 12), соответствующий эффекту плавной трансформации.

ВНИМАНИЕ —

Тип преобразования и его длительность можно задать не только в определении стиля изображения, но и в сценарии.

```
<HTM1_>
<BODY onload = "transformO ">
<IMG ID = "myimg" SRC = "pict.gif" STYLE = "position: absolute ; top:10:
leftrSO; filter:revealtrans">
</BODY>
<SCRIPT>
function transformO {                                  // появление изображения
document . all .myimg.style.visibility = "hidden"  /* делаем изображение
                                                          невидимым */
myimg. filters("revealtrans") .apply()              /* фиксируем исходное
                                                       состояние изображения*/
myimg. style.visibility = "visible"                 // депаем изображение
видимым
myimg. filtersC'revealtrans") .Transi tion = 12    /* указываем тип
                                                        преобразования */
myimg. f iltersC'revealtrans") . play(3)            /* выполняем
                                                        преобразование
                                                        длительностью 3с */
}
</SCRIPT>
</HTML>
```

Очевидно, чтобы добиться исчезновения изображения, необходимо поступить наоборот: сначала сделать его видимым, а затем невидимым.

Теперь рассмотрим более общий случай: трансформацию одного графического объекта в другой. Вы должны понимать, что в отличие от рассмотренного выше случая, эта задача сводится к установке начального и конечного изображений. Это делается путем присвоения нужных значений свойству src объекта, соответствующего изображению.

```
<HTML>
<BODY onload = "transform2() ">
<IMG ID = "myimg" SRC = "pictl.gif" STYLE = "position: absolute; top:10;
left:50; filter: revealtrans(duration = 3, transition = 12)">
</BODY>
<SCRIPT>
function transform2(){                        // появление изображения
myimg. filters ("revealtrans"). apply ()      /* фиксируем исходное
                                                 состояние изображения */
document . all .myimg.src = "pict2.gif"       // заменяем изображение
myimg. filters ("revealtrans") . play ()      /* выполняем
                                                 преобразование */
}
</SCRIPT>
</HTML>
```

В следующем примере зададим трансформацию изображения, которая происходит при щелчке на графическом объекте. При первом щелчке изображение из файла pictl.gif трансформируется в рисунок из файла pict2.gif. При следующем щелчке, наоборот, второй рисунок трансформируется в первый и т. д.

```
<HTML>
<IMG ID = "myimg" onclick = " transformsO" SRC = "pictl.gif" STYLE =
  "position:absolute; top:10; left:50; filter:revealtrans(duration = 3,
  transi tion = 12)">
<SCRIPT>

function transoformB(){                      // преобразование изображения
myimg.filters("revealtrans").apply()        // фиксируем исходное
изображение
if (document.all.myimg.src.indexOf("pictl")!= -1)  /* определяем
                                                      конечное
                                                      изображение */
   document.all.myimg.src = "pict2.gif"
else
   document.all.myimg.src = "pictl.gif"
myimg.filters("revealtrans").play()      // выполняем преобразование
}
</SCRIPT>
</HTML>
```

СВОЙСТВО src содержит полный URL-адрес графического объекта, даже если мы присваиваем ему только имя файла. Поэтому в теле функции transform3() приходится выяснять, входит ли строка с именем файла в URL-адрес, а не проверять, равен ли URL-адрес имени файла.

Рассмотренный выше синтаксис выражений воспринимается браузерами IE4+. Для IE5.5+ можно использовать другой синтаксис, рекомендованный Microsoft. Его особенность состоит в том, что в каскадной таблице стилей задается ссылка на специальный компонент и имя фильтра. Последний пример в этом синтаксисе имеет следующий вид:

```
<HTML>
<STYLE>
#myimg {position:absolute; top:10; left:50;
filter:progid:DXImageTransform.Microsoft.revealtrans(duration = 3,
transition = 12)}
</STYLE>
<IMG ID ="myimg" onclick = "transforms()" SRC = " "earth.gif">
<SCRIPT>

function transforms(){                   // преобразование изображения
my img.fiIters("DXImageTransform.Microsoft.revealtrans ").apply()
                                                 /* фиксируем исходное
                                                    изображение*/
if (document.all.myimg.src.indexOf("earth")!= -1)   /* определяем
                                                       конечное
                                                       изображение*/
   document.all.myimg.src = "sun.gif"
else
   document.all.myimg.src = "earth.gif"
myimg.filters("DXImageTransform.Microsoft.revealtrans").play 0
                                                  /'выполняем
                                                    преобразование */
```

```
}
</SCRIPT>
</HTML>
```

Повороты

В браузерах IE5.5++ возможно применение фильтра basicimage, имеющего множество параметров, с помощью которых изображение можно повернуть на угол, кратный 90°, задать прозрачность, зеркально отразить, определить маску и др. В IE4+ все эти эффекты, за исключением поворотов, создаются отдельными фильтрами. Порядок их применения такой же, как и фильтра alpha, задающего прозрачность. Здесь мы рассмотрим создание эффекта поворота.

Параметр rotation фильтра basic image принимает целочисленные значения: 0 (нет поворота), 1 (90°), 2 (180°), 3 (270°). В примере, приведенном ниже, в результате щелчка на изображении оно поворачивается на угол 90°.

```
<HTML>
<STYLE>
#myimg {filter: progid:DXImageTransform. Mi crosoft. basicimage}
</STYLE>
<IMG ID = "myimg" src = "pict.gif" onclick = "rotor()">
<SCRIPT>
function rotor(){                                    // повороты
var r =
document. all. myimg.fiIters [ "DXImageTransf orm.Microsoft. basicimage"] .
  rotation
 if (r == 3)
   r = 0
else
   r++
document.all.myimg.fiIters ["DXImageTransform.Microsoft. basicimage"] .
  rotation = r
 }
</SCRIPT>
</HTML>
```

Применение нескольких фильтров одновременно

К видимому элементу можно применить несколько различных фильтров одновременно. В следующем примере мы делаем графический объект полупрозрачным с помощью фильтра alpha и трансформируемым с помощью фильтра revealtrans. Полупрозрачным изображение становится сразу при загрузке в браузер, а трансформация происходит при щелчке на нем. Разумеется, вы можете придумать и другой сценарий применения фильтров. Вот вариант кода, реализующий мой сценарий:

```
<HTML>
<IMG ID = "myimg"  onclick = "transforms i()" SRC = "earth.gif"
STYLE = "position.-absolute; top:10; left:50;
filter: revealtrans(duration = 3, transition = 12), alpha(opaci ty = 50)">
<SCRIPT>

function transform3() {                      // преобразование изображения
myimg.filters("revealtrans") .apply()        // фиксируем исходное
                                             // изображение
if (document. all .myimg.src . indexOf ( "pictl") !=-1)  /* определяем
                                                 конечное изображение */
```

```
   document.all.myimg.src = "pict2.gif"
else
   document.all.myimg.src = "pictl.gif"
myimg.fliters("revealtrans").play()            /* выполняем
                                                  преобразование */
}
</SCRIPT>
</HTML>
```

4.1.7. Эффект печати на пишущей машинке

Постепенный вывод на страницу текста (эффект печати на пишущей машинке) можно создать на основе использования метода **setlntervalQ.** Об этом и других методах управления во времени уже говорилось в разделе 2.8.

Ниже приводится HTML-документ с заголовком и текстовой областью, задаваемой тегом <TEXTAREA>. Сценарий в этом документе выводит в текстовую область символы некоторой строки с задержкой 0, 1 с. После завершения вывода всей строки этот процесс останавливается. Функция wrtextQ просто формирует строку, которую требуется вывести в текстовой области в данный момент. Собственно вывод производится путем присвоения значения свойству value текстовой области. Функция wrtextQ передается в виде строкового параметра методу setlntervalQ, который и вызывает ее периодически с интервалом, указанным в миллисекундах, в качестве второго параметра. В примере этот интервал равен 100. Метод setlntervalQ возвращает целочисленный идентификатор запущенного процесса, который мы сохраняем в переменной **xinterval.** Это значение передается методу **clearlntervalQ** как параметр, чтобы завершить процесс периодического вызова функции **wrtextQ,** когда «напечатается» вся строка.

```
<HTML>
<H1>Моя веб-страница</H1>
<TEXTAREA ID = "mytext" ROWS = 8 COLS = 25></TEXTAREA>
<SCRIPT>
var mystr = "Привет, мои друзья! Рад вам сообщить приятное известие"
var astr = mystr.split("")          // разбиваем строку на массив
                                    // символов
var typestr = ""
var i = 0
var xinterval = setlnterval ("wrtext () " , 100) /* периодический вызов
                                                   функции wrtext() */

function wrtext(){  // вызывается с помощью метода setlnterval()
if (i < astr .length){
   typestr+= astr[i]                    // выводимая строка
   document.all.mytext.value = typestr  // вывод строки
   i++
}else
   clearlnterval(xinterval)             // прекращаем вывод текста
}
</SCRIPT>
</HTML>
```

Другие примеры использования метода setlntervalQ вы найдете в следующем разделе.

4.2. Движение элементов

Перемещение видимых элементов HTML-документа в окне браузера основано на изменении значений параметров позиционирования top и left таблицы стилей. Вы можете указать эти параметры в атрибуте STYLE или в теге <STYLE> для тех тегов, которые задают видимые элементы (изображения, тексты, ссылки, кнопки и т. п.). Затем остается только определить в сценарии способ изменения параметров координат top и left для того или иного элемента. Можно заставить элемент перемещаться постоянно, в течение заданного времени или в ответ на события (например, по щелчку кнопкой мыши, при наведении указателя мыши на элемент и т. п.).

ВНИМАНИЕ

Оператор цикла для организации расчета координат обычно не применяется, поскольку, пока выполняется цикл, другие выражения сценария не работают.

Чтобы распараллелить вычислительные процессы, используют методы setlntervalQ и setTimerQ, описанные в разделе 2.8.

4.2.1. Движение по заданной траектории

Схема сценария, осуществляющего непрерывное перемещение видимого элемента документа, имеет следующий вид:

```
function init_move() {          // инициализация движения
...                              // подготовка к запуску функции move()
setlnterval("move()", задержка)
}
function move(){
      ...                        /* изменение координат top и left
                                    стиля перемещаемого элемента */
}
init_move()                      // вызов функции для перемещения элемента
```

Таким образом, мы создаем две функции, имена которых могут быть произвольными. Первая функция, init_move(), осуществляет подготовку исходных данных и вызывает метод setlntervalQ с указанием в качестве первого параметра имени второй функции move() в кавычках. Вообще говоря, первый параметр метода setlnterva() является строкой, содержащей выражение, которое должно выполняться периодически во времени. Вторая функция, move(), изменяет координаты элемента. Поскольку метод setlntervalQ вызывает функцию move() периодически через заданное количество миллисекунд, то координаты элемента изменяются постоянно. При этом создается эффект движения. Скорость и плавность движения зависят от величин приращения координат (в функции moveQ) и временной задержки (второго параметра метода setlntervalQ). Чтобы начать перемещение элемента, необходимо просто вызвать первую функцию, init_move().

Линейное движение

Рассмотрим в качестве примера сценарий линейного перемещения изображения, то есть движения по прямой линии.

```
<HTML>
<IMG ID="myimg" SRC="pict.gif " STYLE="position: absolute; top:10;left:20">
<SCRIPT>
function init_move() {
dx = 8                                // приращение по y
dy = 3                                // приращение по x

setInterval("move()", 200)            // периодический вызов функции
move ()
}

function move() {                     // изменение координат изображения
/* Текущие координаты: */
var y = parselnt (document.all.myimg.style.top)
var x = parselnt (document.all.myimg. style.left)
/* Новые координаты: */
document . all.myimg.style . top = y + dy
document.all. myimg.style.left = x + dx
}

init_move()                           // начинаем движение
</SCRIPT>
</HTML>
```

Обратите внимание, что переменные dx и dy являются глобальными и поэтому видны в функции move(). Если бы мы определили их в init_move() с помощью ключевого слова var, то они стали бы локальными и недоступными в moveQ. Заметьте также, что значения параметров позиционирования в таблице стилей имеют строковый тип. Поэтому нам потребовалось применение функции parselntQ для приведения их к числовому типу.

Теперь усовершенствуем приведенный выше код: сделаем так, чтобы идентификатор перемещаемого объекта, а также приращения координат и временную задержку можно было передавать функции init_move() в качестве параметров. В результате мы получим универсальную функцию линейного перемещения любого видимого элемента. Вот вариант соответствующего кода:

```
function init_move(xid, dx, dy) {
var prmstr = "'" + xid + "'," +dx + "," + dy     /* строка параметров
                                                    для move() */
prmstr = "move(" + prmstr + ")"
setlnterval(prmstr, 200)                         /* периодический вызов
                                                    функции move() */
}

function move(xid, dx , dy) {
y = parselnt(document. all[xid].style.top)
x = parselnt(document. all[xid] . style. left)
document.all.myimg.style.top = y + dy
document.all.myimg.style. left = x + dx
)

init_move("myimg", 10, 5)                        // пример вызова функции
```

Обратите внимание, как методу setlntervalQ передается указание на функцию moveQ с параметрами. В явном виде параметры функции, указанной в вызове метода setlntervalQ, передавать нельзя, поскольку первый параметр этого метода имеет строковый тип. Поэтому мы и формируем то, что ему требуется, — строку.

Остановка движения

Рассмотренные выше сценарии обеспечивают непрерывное перемещение элемента документа. Поскольку это перемещение осуществляется по прямой **линии,** то рано или поздно элемент выйдет за пределы окна браузера. Разумеется, можно усовершенствовать функцию **moveQ** таким образом, чтобы она удерживала элемент в некоторой области окна браузера, изменяя в случае необходимости, например, знак приращений координат. Однако может потребоваться, чтобы элемент пересек окно браузера и больше не возвращался. Тогда полезно остановить движение уже не видимого элемента. Для этого служит метод **clearlnterval(),** единственным параметром которого является целочисленный идентификатор, возвращаемый методом **setlntervalQ.**

Чтобы иметь возможность прекратить движение элемента, следует сохранить значение, возвращаемое методом setlntervalQ, в глобальной переменной, а затем использовать его в качестве параметра метода **clearlntervalQ** в теле функции moveQ. Напомню, что moveQ — функция, имя которой передается в качестве первого параметра методу setlntervalQ. Сценарий с запуском и остановкой движения может выглядеть, например, следующим образом:

```
function init_move(xid, dx, dy) {
var prmstr = "'" + xid + "'," +dx + "," + dy          /* строка
                                                        параметров
                                                        для move() */
prmstr = "move(" + prmstr + ")"
id_move = setlnterval (prmstr , 200)                    /* сохраняем
                                                        идентификатор
                                                        движения */
}

function move(xid, dx , dy) {
y = parselnt(document.all[xid].style.top)
x = parselnt(document .all[xid] . style. left)
document.all.myimg.style.top = y + dy
document.all.myimg.style.left = x + dx
if (parselnt(document. all[xid] .style.left) > 350)  /* остановка
                                                        по условию */
    clearlnterval(id_move)
}

init_move("myimg", 10, 5)                               // начинаем движение
```

В этом примере движение остановится, как только горизонтальная координата элемента превысит 350 пикселов.

Движение по эллипсу

Теперь рассмотрим организацию движения по замкнутой траектории. Для примера возьмем эллипс, поскольку его легко модифицировать. Движение по эллипсу задается несколькими параметрами, такими как большая и малая полуоси, положение центра и угол поворота относительно горизонтали, угловая скорость перемещения и др. (рис 4.6). В частном случае, когда одна из полуосей эллипса равна нулю, траектория вырождается в прямую линию. При этом движение будет происходить то в одну, то в другую сторону. Очевидно, в случае равенства полуосей траектория приобретает вид окружности.

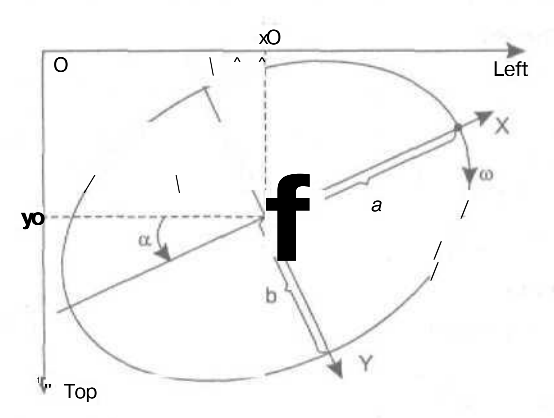


Рис. 4.6. Эллиптическая траектория

В листинге 4.5 представлены определения двух функций, задающих движение по эллипсу. Функция ellipseQ является функцией инициализации движения, a moveQ — функция, передаваемая методу setlntervalQ.

Листинг 4.5. Определения двух функций, задающих движение по эллипсу

```
function ellipse(xid, alpha, a, b, omega, x0, y0, ztime.dt) {
/* Движение по эллипсу
Параметры:
xid - id движущегося объекта, строка
alpha - угол поворота эллипса, град.
x0 - x-координата центра эллипса, пикселы
y0 - y-координата центра эллипса, пикселы
a - большая полуось эллипса, пикселы
b - малая полуось эллипса, пикселы
omega - угловая скорость, град/с; знак задает направление вращения
ztime - начальная фаза, град.
dt - временная задержка, с
*/

// проверка наличия параметров:
if (lalpha) alpha = 0
тf (!al) a = 0
if (!b) b = 0
if (Iomega) omega = 0
if ; !x0) x0 = 0
it ; !y0) y0 = 0
if (!ztime) ztime = 0
if (!dt) dt = 0

var t = -ztime              /* чтобы начальное значение было 0,
                            поскольку в move() уже есть приращение */
setlnterval("move('" + xid + "'," + alpha + "," + a +"," + b + "," +
  omega + "," + x0 + "," + y0 + "," + ztime + "," + dt + ")",
  ztime*1000)      /'многократный вызов moveO с интервалом ztime, мс */
}

function move(xid, alpha, a, b, omega, x0, y0, ztime, dt) {
                            /* пересчет координат,
                            вызывается из ellipseO */
```

продолжение

Листинг 4.5 (продолжение)

```
t+= ztime
/* x,y - координаты в собственной системе координат */
var x = a*Math.cos((omega*t + dt)*Math.PI/180)
var y = b*Math.sin((omega*t + dt)*Math.PI/180)
var as = Math.sin(alpha*Math.PI/180)
var ac = Math.cos(alpha*Math.PI/180)
document.all[xid].style.top = -x*as + y*ac + y0
document.all[xid] . style. left = x*ac + y*as + x0
}
```

Ниже приведен HTML-код, содержащий сценарий, который загружает два изображения и заставляет их двигаться по эллиптическим траекториям:

```
<HTML>
<SCRIPT>
function ellipse(xid, alpha, a. b, omega, x0, y0, ztime,dt) {
. . .                                      // код определения функции
}
function move(xid, alpha, a, b, omega, x0, y0, ztime, dt) {
, . .                                      // код определения функции
}

var xstr = '<IMG ID = "myl" SRC = "pictl.gif"><IMG ID = "my2" SRC =
  "pict2 .gif">'
document.wri te(xstr)

ellipse("myl",. 60, 100, 30, 10, 170, 300, 0.2, 0)
ellipse("my2",. 80, 100, 130, 30,140, 300, 0.2, 0)
</SCRIPT>
</HTML>
```

Если сохранить код с определением функций **ellipseQ и moveQ** в файле, например **ellipse.js,** то рассмотренный выше HTML-код можно переписать в следующем виде:

```
<HTM1_>
<SCRIPT SRC = "ellipse. js"></SCRIPT>
<SCRIPT>
var xstr = '<IMGID = "myl" SRC = "pictl. gi f 'xIMG ID = "myl" SRC =
  "pictl.gif">'
document.write(xstr)
ellipseC'myl", 60, 100, 30. 10, 170, 300, 0.2, 0)
ellipse("my2", 80, 100, 130, 30,140, 300, 0.2, 0)
</SCRIPT>
</HTML>
```

Движение по произвольной кривой

Рассмотрим задачу организации движения видимого элемента по произвольной кривой, заданной выражением с одной переменной. Точнее, мы хотим указать функции движения в качестве параметров не одно, а два выражения, которые описывают изменения вертикальной и горизонтальной координат элемента. Эти выражения будут содержать одну переменную, которую мы обозначим через x — строчной латинской буквой. Переменную x можно интерпретировать как независимый параметр движения (например, время). С помощью встроенной функции **evalQ** можно вычислить значения этих выражений при конкретном значении переменной x и присвоить их параметрам left и top таблицы стилей перемещаемого элемента. Функция (пусть это будет moveQ), которая все это выполняет, переда-

ется в качестве первого параметра методу setlntervalQ, который периодически вызывает ее через заданный интервал времени.

Итак, функция инициализации движения **curvemoveQ** будет принимать три строковых и один числовой параметр. Строковые параметры содержат соответственно значение идентификатора ID перемещаемого элемента, выражение для вертикальной координаты и выражение для горизонтальной координаты. Числовой параметр определяет период времени, через который координаты элемента пересчитываются. Ниже приводятся определения функций curvemoveQ и move():

```
function curvemove(xid, yexpr, xexpr, ztime) {
/* Движение по произвольной кривой. Версия 0.
   Параметры:
   xid - id движущегося объекта, строка
   yexpr - выражение для вертикальной координаты
   xexpr - выражение для горизонтальной координаты
   ztime - интервал времени между вызовами функции move(), мс
*/

if ( !xid) return null
if (lyexpr) yexpr = "x"
if (,xexpr) xexpr = "x"
if (Iztime) ztime = 100          // интервал времени, мс
x=0                               /* глобальная переменная,
                                     входящая в выражения yexpr и xexpr */
setlnterval("move('" + xid + "','" + yexpr + "','"+ xexpr + "')",
  ztime)
}

function move(xid, yexpr, xexpr) {
x++
document.all[xid].style.top = eval(yexpr)
document.all[xid].style.left = eval(xexpr)
}
```

Обратите внимание, что переменная x в функции curvemove() является глобальной и поэтому доступна в функции move(). Чтобы сделать переменную x глобальной, мы просто не использовали ключевое слово var перед первым ее появлением в коде.

Исходное позиционирование перемещаемого элемента с помощью таблицы стилей не играет особой роли, поскольку при вызове функции curvemoveQ элемент помещается в точку с координатами, равными значениям выражений xexpr и yexpr, вычисленным при x = 0. Поэтому начальное позиционирование следует задавать с помощью соответствующего выбора этих выражений. Ниже приведен пример HTML- документа с движущимся изображением:

```
<HTML>
<IMG ID = "myimg" SRC = "pictl.gif" STYLE = "position:absolute">
<SCRIPT>
function curvemove(xid, yexpr, xexpr, ztime) {
                                         // код определения функции
}
function move(xid, yexpr, xexpr) {       // код определения функции
}
```

```
curvemoveC'myimg" , "100 + 50*Math . sin(0. 03*x)" , "50+x", 100)
</SCRIPT>
</HTML>
```

В этом примере изображение будет перемещаться по синусоиде с амплитудой 50 пикселов и горизонтальной скоростью 10 пикселов в секунду. Начальные координаты графического объекта равны 100 и 50 пикселов по вертикали и горизонтали соответственно.

В рассмотренных выше функциях curvemoveQ и moveQ в выражениях, описывающих движение, в качестве аргумента можно использовать только символ x. Ниже приведены модификации этих функций, позволяющие использовать произвольные имена для аргумента, лишь бы они не совпадали с уже применяемыми для других целей:

```
function curvemove(xid, yexpr, xexpr, ztime, namevar) {
/* Движение по произвольной кривой. Версия 1.
   Параметры:
   xid - id движущегося объекта, строка
   yexpr - выражение для вертикальной координаты
   xexpr - выражение для горизонтальной координаты
   ztime - интервал времени между вызовами функции move(), мс
   namevar - имя аргумента в выражениях yexpr и xexpr
*/

if (!xid) return null '
if (lyexpr) yexpr = "x"
if (Ixexpr) xexpr = "x"
if (Izti me) ztime = 100              // интервал времени, мс
if (Inamevar) namevar = "x"
eval(namevar + "= 0")                 /* глобальная переменная, входящая
                                         в выражения yexpr и xexpr */
setInterval("move(' " + xid + "', '" + yexpr + "', " + xexpr + ", '"+
namevar + " ' ) " , ztime)
}

function move(xid, yexpr, xexpr, namevar) {
eval(namevar + "++")
document . all[xid] . style.top = eval(yexpr)
document. all [xid]. style.left = eval(xexpr)
\
```

Суть модификации кода заключается в том, что имя аргумента теперь передается функциям в качестве строкового параметра namevar. Чтобы создать переменную с именем, указанным в значении переменной namevar, и присвоить ей некоторое значение, необходимо создать строку с оператором присвоения и передать ее функции eval(). Теперь можно использовать, например, такие вызовы функции:

```
curvemoveC'myimgl" , "100 + 100*Math. si n(0.05*w)/(0. 05*w)" , "50+w", 50, "w")
curvemove("myimg2", " , "100+ 2*time)", "570 - time", 100,"w")
```

Обратите особое внимание на использованный выше прием создания переменной. Сначала имя этой переменной сохраняется в виде строкового значения в другой переменной, затем она используется при генерации строки, содержащей некоторое выражение. Далее строка с выражением передается в качестве параметра функции evalQ, которая выполняет данное выражение. Этот прием лежит в основе создания программ, которые пишут программы.

4.2.2. Перемещение мышью

Рассмотрим задачу перемещения видимых элементов HTML-документа с помощью мыши. Такими элементами могут быть графические объекты, кнопки, текстовые области, таблицы, плавающие фреймы и др. Здесь мы займемся перемещением изображений, текстовых областей и плавающих фреймов.

Перемещение графических объектов

Перемещать мышью изображения можно различными способами. Изучим один из них: пользователь пытается перетащить мышью изображение; затем он должен отпустить кнопку мыши и переместить указатель в нужное место (при этом он может удерживать или не удерживать кнопку мыши в нажатом положении); остановившись в нужном месте, пользователь отпускает кнопку мыши или щелкает ею, чтобы прекратить перемещение изображения. В листинге 4.6 приведен код, реализующий этот алгоритм.

x

Листинг 4.6. Код перемещения изображения мышью

```
<HTM1_>
<HEAO><T1T1_E>Перемещаемое изображение</T1T|_E></HEAO>
<BODY id = "mybody">
<IMG ID="myimg" SRC = 'pict.gif ondragstart = "dragO " style =
  "position:absolute; top:10; left:10">
</BODY>
<SCRIPT>
flag = false                          // нельзя перемещать
var id_img = ""

function drag() {
flag = mag
id_img = event.srcElement.id
   }

function mybody.onmousemove(){
if (flag){                            // если можно перемещать
document.all[id_img].style.top=event.clientY
document.all [id_img].style.left = event.clientX
}
}

function mybody .onmouseupO{
flag = false                          // нельзя перемещать
}
</SCRIPT>
</HTML>
```

Здесь функция **dragQ,** обрабатывающая событие ondragstart (попытка перетаскивания), устанавливает переменную-триггер flag и выясняет, кто инициатор события. Значение переменной flag позволяет определить, можно или нельзя перемещать элемент. В данном примере инициатором события может быть только один элемент, но мы готовимся к более общему случаю нескольких перемещаемых элементов. Этот случай будет рассмотрен позже. Обратите внимание, что события onmousemove (перемещениеуказателя мыши) иonmouseup (кнопка мыши отпущена) получает не изображение, а объект тела документа mybody. Мы не исполь-

зовали событие onclick для изображения, зарезервировав его для других целей, например для перехода по ссылке. Заметьте также, что в функции mybody. onmousemove() при обращении к объекту изображения его идентификатор передается через переменную id_img. Поэтому мы используем запись document.all[id_img], а не document.all.idjmg.

Теперь рассмотрим более сложную задачу, используя для ее решения приемы, применявшиеся ранее. Пусть документ содержит несколько графических объектов, которые можно перемещать мышью. В нашем случае это изображения парусников. Все множество кораблей разбито на две эскадры: ту (моя) и other (чужая). Визуально они отличаются лишь ориентацией кораблей и флагом. Размер корабля зависит от значения его вертикальной координаты top: чем ее значение больше, тем больше размеры рисунка. Таким способом мы создаем эффект перспективы (ближе-дальше). Чем ниже расположено изображение, тем больше **его** размеры. Далее, если к какому-нибудь кораблю приблизились на достаточное расстояние чужие корабли в достаточном количестве, то какой-то из них погибает (становится невидимым). Для решения, какой из сблизившихся кораблей должен погибнуть, используется жребий (датчик случайных чисел Math.random()). Это — прототип алгоритма некоторой игры типа «Морской бой» (рис. 4.7). Впрочем, нам важен не сюжет игры, а приемы программирования.

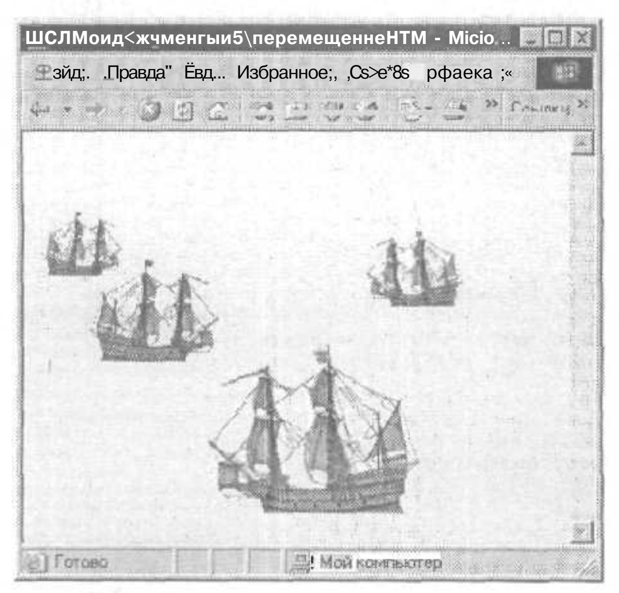


Рис. 4.7. Игра «Морской бой»

В рассматриваемом примере используются два графических файла с изображением кораблей: один для своих (shipl.gif), а другой — для чужих (ship2.gif). В HTML-документе можно расположить необходимое количество графических элементов с помощью тегов, написав их вручную:

```
<IMG ID = "my1" SRC = "shipl.gif" ondragstart = "dblclO" STYLE =
"position:absolute; top:10;left:10; visibility:visible">
```

```
<IMG ID = "my2" SRC = "shipl.gif" ondragstart = "dblcl()"STYLE =
"position: absolute; top:10; left:10; visibility:visible">
. . .
<IMG ID = "otherl" SRC = "ship2.gif" ondragstart = "dblclO" STYLE =
"position: absolute ; top:10; left:10; visibility:visible">
<IMG ID = "other2" SRC="ship2.gif" ondragstart="dblcl()" STYLE =
"position: absolute; top:10; left:10; visibility:visible">
. . .
```

Однако проще создать сценарий, выполняющий это автоматически. Начальные координаты кораблей зададим с помощью генератора случайных чисел (листинг 4.7).

Листинг 4.7. Код сценария для игры «Морской бой»

```
<HTML>
<HEAD><TITLEПеремещаемые изображения</T1T1E></HEA0>

<BODY id="mybody" background="blue.gif"></BODY>
<SCRIPT>
var imgstr=""                    // строка тегов, формирующих рисунки
var Nmy=3                        // количество своих
var Nother=3                     // количество чужих
/* Формирование изображений */
for (i = 1; i <= Nmy; i++) {
imgstr+= '<IMGID="my' + i + '" src="ship1.gif" ondragstart="drag()"
style="position:absolute;top:' + 200*Math.random() + ';left:' +
  200*Math.randomC) + ';visibility:visible">'
}
for (i = 1; i <= Mother; i++){
imgstr+= '<IMG ID="other' + i + '" src="ship2.gif" ondragstart="drag()"
style="position:absolute;top:' + 200*Math.random() + ';left:' +
  200*Math.random0 + ';visibility:visible">'
}
document.write(imgstr)
                                 // запись в документ

resize0                          // установка размеров изображений
var flag = false
var id_img = ""

function drag()
{
flag = Iflag
id_img = event.srcElement.id     // id элемента, который надо перемещать
}

function mybody.onmousemove0{
if (flag){
   document.all[id_img}.style.top = event.clientY
   document.all[id_img].style.left = event.clientX
   resize0                       // установка размеров изображений
}
}

function mybody.onmouseupO {
flag = false
action()                         // действия в создавшемся положении
}

function action() {              // действия
var s, xO, y0, xl, yl, vl, v2, v3. xid, yid, d, z
```

продолжение

Листинг 4.7 *(продолжение)*

```
var dmin = 20, smin =2          /* минимальное расстояние и
                                   количество чужих */
for (i = 0; i < document.images.length; i++){
  s = 0
  x0 = parselnt (document. images[i].style. left)
  y0 = parselnt(document. images[i].style. top)
  for ( j=0; ] < document.images.length; ]++){
   vl = document. images [i].style.visibility == "visible"
   v2 = document . images!j] . style.visibility == "visible"
   xid = document. images [i].id.substrCO,2)
   yid = document . images []] .id. substr (0,2)
   v3 = xid. localeCompare(yid)  != 0    // чужой
      if (( i != j)&&vl&&v2&&v3){
         xl = parselnt(document. images [j].style. left)
         yl = parselnt(document. images[j].style. top)
         d = Math.sqrt((x0 - xl)*(x0 - xl) + (y0 - yl)*(y0 - yl))

      if (d <= dmin) s++              // очень близко
      if (s >= smax) {
         if(Math.random()<0. 5) z = i   // кто погибнет
         else z = j
         document. images [z] . style.visibility = "hidden"
         break
      }
    }
}
}
}

function resize(){                    // установка размеров изображений
var y, size
for (i = 0; i < document . images . length; i++){
   y = parselnt(document. images [i].style.top)
   size = Math. min(x, 180)
   size = Math.max(size, 15)
   document . images[i] .style.width = size
   document. images [i]. style. height = 0.8*size
}
}
</SCRIPT>
</HTML>
```

Как нетрудно заметить, главная часть семантики игры обслуживается функцией **actionQ**. Вы можете пофантазировать и изменить ее по своему усмотрению. **Я** же обратил внимание на пространственные свойства театра военных действий. **В** нашем случае не исключены ситуации, когда кораблик, расположенный выше (значит, дальше), накрывает кораблик, расположенный ниже (значит, находится на переднем плане). Иначе говоря, наш способ создания эффекта перспективы плохо работает в случае перекрытия изображений. Поправить это можно путем изменения параметра **z-Index** стилей элементов в функции resize(), поставив это свойство в прямую зависимость от значения вертикальной координаты top. Элемент с большим индексом будет находиться над элементом с меньшим индексом. Модифицированный код функции **resizeQ** выглядит следующим образом:

```
function resize () {
var y, size
```

```
for (i =0; i < document. images . length ; i++){
   y = parselnt(document.images[i].style.top)
   size = Math.min(y, 180)
size = Math.max(size, 15)
document. images [i ]. style.width = size
document . images [1 ]. style . height = 0.8*size
document.images[i].style.zlndex = x
}
}
```

Обратите внимание, как в JavaScript происходит обращение к свойству z-index таблицы стилей.

Перемещение текстовых областей

В листинге 4.8 приводится пример HTML-документа, в котором можно перемещать текстовые области, созданные с помощью тегов <TEXTAREA>. При этом размеры области и шрифта текста определяются в ней в зависимости от значения вертикальной координаты. Так создается эффект перспективы (ближе-дальше). Чем выше, тем дальше, и наоборот (рис. 4.8). Чтобы приблизить к себе текстовую область, необходимо просто переместить ее вниз. Вы можете «сложить в стопку» текстовые области, удалить их от себя или, наоборот, приблизить так, чтобы текст стал разборчивым. В отличие от примера с изображениями, здесь разрешение на перемещение происходит по двойному щелчку на текстовой области. Обратите внимание на то, как в функции resizetextQ определяется элемент, размеры которого следует изменить.

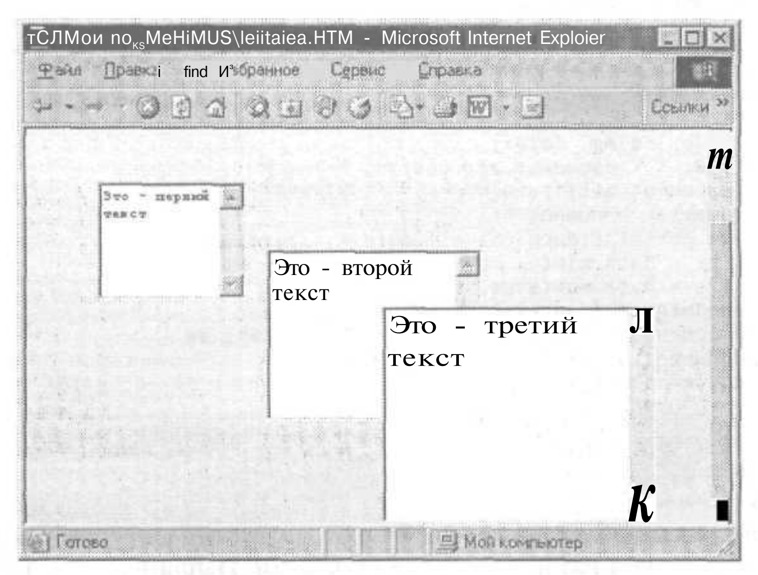


Рис. 4.8. Перемещаемые текстовые области

Листинг 4.8. Перемещение текстовых областей, созданных с помощью тегов <TEXTAREA>

```
<HTML>
<HEAO><T1T1E>Перемещаемые текстовые области</T1T1E></HEAO>
<BODY id="mybody" background="blue.gif">
```

продолжение

Листинг 4.8 *(продолжение)*

```
<TEXTAREA ID="t1" ondblclick=""drag()" STYLE = "position:absolute;
  top:10; left:10;fon-size:large">3To - первый текст</TEXTAREA>
<TEXTAREA ID="t2" ondblclick=""drag()" STYLE = "position:absolute;
  top:100; left:150">Это - второй текст</TEXTAREA>
<TEXTAREAi ID="t3" ondblclick=""drag()" STYLE = "position:absolute;
  top:150; left:250">Это - третий ТеКСТ</TE)XTAREA>
</BODY>
<SCRIPT>
resizetext()                              // установка размеров текстовых
                                          // областей
var flag = false
var id_img = ""

function drag()
{
flag = !flag
id_img = event.srcElement.id              // id элемента, который надо
                                          // перемещать
1

function mybody . onmousemove () {
if  (flag){
   document . all [id_img] .style,top = event . clientY
   document.all [id_img].style.left = event.clientX
   resizetext0                    // установка размеров текстовых областей
}
}

function mybody . onmouseupO {
flag = false
}

function resizetext () {                 // установка размеров текстовых
                                         // областей
vary, size, idimg, idtext
for (i =0; i < document . all . length; i++) {
   if (document.all [i ] .tagName== 'TEXTAREA') {
      idtext = document . all[i].id
      y = parselnt(document.all [idtext].style.top)
      size = Math.min(y, 800)
      size = Math.max (size, 60)
      document . all [idtext] . style.width = size
      document.all [idtext] . style.height = 0.8*size
      document.all[idtext].style.zlndex = y
      document.all [idtext].style.fontSize = Math.max(2, y/10)
   }
}
}
</SCRIPT>
</HTML>
```

В принципе, ничто не мешает вам создать более сложный алгоритм для функции resizetext() изменения размеров текстовой области. Например, можно дополнительно варьировать цвет фона и шрифта.

Перемещение плавающих фреймов

Если рассмотренный выше код немного модифицировать, то можно получить документ со множеством перемещаемых плавающих фреймов, в которые загруже-

ны другие HTML-документы (рис. 4.9). Однако в этом случае возникает небольшая проблема: как передать плавающему фрейму сигнал, что мы собираемся его перемещать. Дело в том, что сигналы о нажатии кнопки мыши, щелчках и попытках перетаскивания при наведении указателя мыши на фрейм получает не он, а элементы загруженного в него документа. Я выбрал следующий способ решения данной задачи.

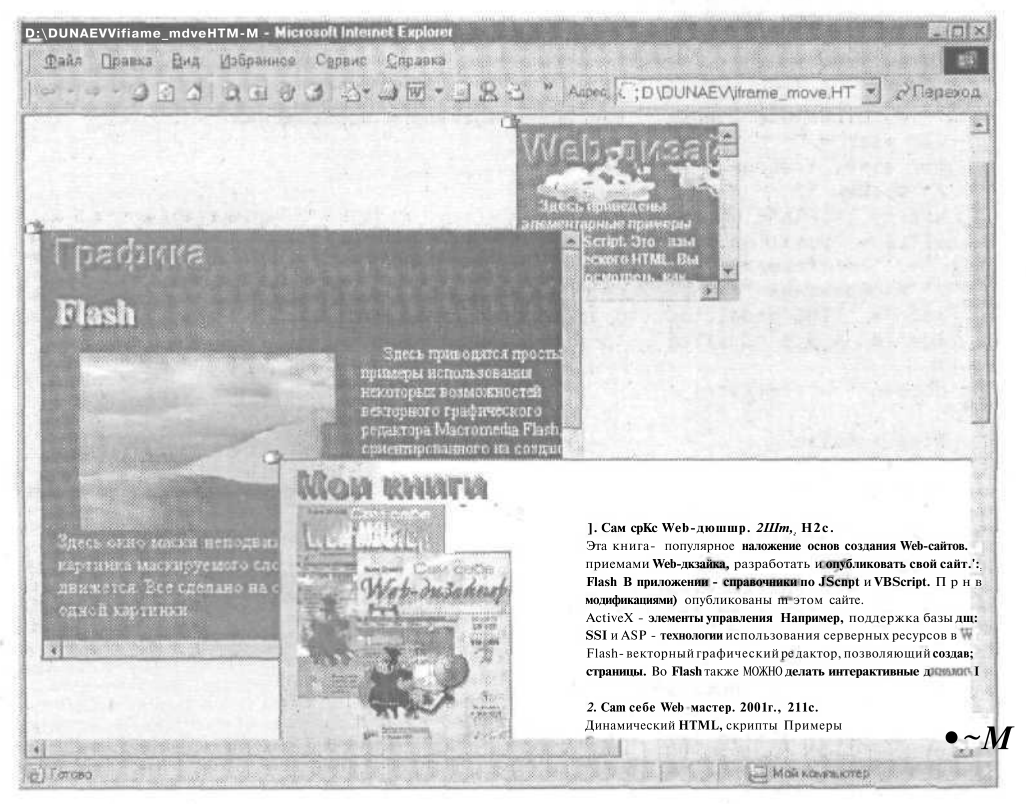


Рис. 4.9. В перемещаемые фреймы загружены страницы сайта

Во-первых, рядом с каждым фреймом поместим маленькое изображение, которое будет воспринимать событие, интерпретируемое как разрешение на перемещение фрейма. Я использовал изображение руки, расположенное около верхнего левого угла фрейма. Во-вторых, в функции изменения положения напишем код, который изменяет координаты, размеры и индекс фрейма, а также графического объекта. Таким образом, изображение будет получать событие, а обработчик этого события будет изменять параметры и изображения, и фрейма.

Напомним, что плавающие фреймы создаются с помощью контейнерного тега <IFRAME>. В сценарии (листинг 4.9) мы сначала сформируем массив с адресами документов (файлов), загружаемых во фреймы. Затем сформируем строку тегов, описывающих фреймы и графические изображения, и запишем ее в HTML-документ. Функция, производящая вычисление положения и размеров фреймов и изображений, в этом примере называется resizeframeQ.

Листинг 4.9. Код для перемещения плавающих фреймов

```
<HTML>
<HEAD><TITLE>Перемещаемые фреймык/TITLE></HEAD>
<BODY ID = "mybody" background = "blue.gif">
</BODY>
<SCRIPT>
/* Массив адресов документов */
var ahref=new ArrayO
ahref[0] = "graphic . htm"
ahref[1] = "examples.htm"
ahref[2] = "mybook.htm"
/* Формирование строки с описанием фреймов и изображений */
var xstr = ""
for (i=0; i<ahref.length;i++) {
/* Фреймы */
xstr+= '<IFRAME width=200 ID="fr' + i + ' " SRC = "' +ahref [i ] + '
STYLE = "position: absolute; top:' + (5 + 25*i) + ' ; left:' + (10 + 30*i)
  + ' "></iframe>'
/* Изображения */
xstr+= '<IMG ondblclick="drag()" width=16 height=14 SRC = "pyKa.gif"
id="im' + i + '" STYLE = "position: absolute"> '
}
document.write(xstr)                          // запись в документ

flag = false
var id_img = ""
resizef rameO                                 // установка размеров фреймов

function drag(){
flag=!flag
id_img = event.srcElement. id
id_img = "fr" + id_img.substr(2)
} "

function mybody.onmousemoveO{
if (flag){
   document.all [id_img].style.top= event. clientY
   document.all[id_img].style.left = event.clientX
   resizeframeO
}
}

function mybody . onmouseupO {
flag = false
}

function resizeframe(){                       // установка размеров фреймов
var y. size, idimg, idfr
for (i = 0 ; i < ahref. length; i++){
   /* Фреймы */
   idfr = "fr" + i                            //id фрейма
   y = parselnt(document.all[idfr].style.top)
   size = Math.min(y,800)
   size = Math .max(60,size)
   size = size*3
   document.all[idfr].style.width = size
   document.all [idfr] . style .height = 0.8*size
```

```
    document.all[idfr].style.zlndex = y
    /* Изображения */
    idimg = "lm" + i                     //id изображения
    document.all[idimg].style.top = y-5
    document.all[idimg].style.left =
      parselnt(document.all[idfr].style.left) -12
    document.all[idimg].style.zlndex=document.all[idfr].style.zlndex
  }
  |
  </SCRIPT>
  </HTML>
```

Обратите внимание, что перемещение фрейма приостанавливается, когда указатель мыши находится над фреймом. Поэтому для перемещения фрейма указатель мыши должен двигаться над участками окна, не занятыми фреймами. Это небольшое неудобство вызвано тем, что фрейм маскирует тело (элемент <BODY>) документа исходного окна и обработчик rnybody.onmousemove() не срабатывает. Маскирующий эффект появляется из-за того, что фрейм является разновидностью окна. В случае с текстовыми областями этот эффект не наблюдается, поскольку текстовая область — просто элемент документа текущего окна, а не отдельное окно.

Мне кажется, что при надлежащем использовании техники перемещения текстовых областей, плавающих фреймов и других элементов (например, таблиц) можно разработать новый стиль дизайна HTML-документов (веб-страниц). Его суть заключается в создании псевдотрехмерного пространства, в которое погружены элементы документа. Эти элементы представлены в документе не ссылками, а своими уменьшенными копиями, расположенными где-то «вдали». Их всегда можно перетащить на передний план, чтобы их было лучше видно, или, наоборот, отдалить (отложить в долгий ящик). Иначе говоря, можно сделать документ похожим не на газетный лист, а на ландшафт, наблюдаемый с высоты птичьего полета.

4.3. Рисование линий

В JavaScript нет специальных встроенных средств для рисования произвольных линий. Если вам потребуется отобразить в окне браузера прямоугольник или горизонтальную линию, то для этого можно воспользоваться HTML-тегами <TABLE> и <HR> соответственно. А как быть, если нужны наклонная прямая, круг или кривая, заданная уравнением? Например, как изобразить график некоторой зависимости в виде кривой, а не последовательности столбиков?

Идея решения этой задачи довольно проста. Нужно вывести на экран изображение размером 1x1 пиксел, залитое цветом, отличающимся от цвета фона. Это изображение следует разместить несколько раз в соответствии с координатами, которые задаются параметрами позиционирования top и left атрибута STYLE тега <IMG>. С помощью сценария можно сформировать строку, содержащую теги <IMG> с необходимыми атрибутами, а затем записать ее в документ методом write().

В этом разделе мы рассмотрим сначала задачу рисования прямой линии, а затем — произвольной кривой, заданной выражением с одним аргументом.

4.3.1. Прямая линия

Прежде всего выясним, как нарисовать точку. Из множества таких точек будет состоять любая кривая, которую нам нужно отобразить в окне браузера. В HTML для этого можно использовать следующий тег:

```
<IMG SRC = "point.bmp" STYLE = "position:absolute;top:y;left:x">
```

Здесь point.bmp — имя графического файла, содержащего один пиксел; y, x — числа, указывающие положение графического файла в пикселах. Изображение точки размером 1 x 1 пиксел можно создать в любом графическом редакторе. Из соображений экономии его лучше всего сохранить в файле формата BMP, а не JPEG или GIF (при малых размерах изображения алгоритмы сжатия неэффективны).

Чтобы задать размеры отображения точки на экране, следует использовать атрибуты WIDTH и HEIGHT (ширина и высота):

```
<IMG SRC = "point.bmp" STYLE = "position:absolute;top:y;left:x" WIDTH =
  n HEIGHT = n>
```

Одинаковые значения атрибутов WIDTH и HEIGHT задают представление точки в виде квадрата размером nх n пикселов. При этом точка с исходными размерами 1x1 пиксел просто растягивается. Таким образом, мы можем задать отображаемые размеры (масштаб) одной точки, а следовательно, и определить толщину линии.

Теперь рассмотрим начальную версию функции, которая рисует прямую линию с заданными координатами x1, y1, x2, y2 ее начала и конца. Кроме координат функция будет принимать числовой параметр n, указывающий толщину линии. Вот вариант определения функции:

```
function line(x1, y1, x2, y2 , n){
/* Версия 0 */
/* x1, y1 - начало линии x2 , y2 - конец линии n - толщина линии */

var clinewidth = " WIDTH=" + n + "HEIGHT=" + n    /* строка для учета
                                                     толщины */
var xstr = ""   // строка тегов для записи в HTML-документ
var xstr0 = '<IMG SRC="point.bmp"' + clinewidth + '
STYLE="position:absolute; '
var k = (y2 - y1)/(x2 - x1)          // коэффициент наклона линии
var x = x1                           // начальное значение координаты x
/* Формирование строки, содержащей теги <IMG...> */
while (x <= x2) {
   xstr += xstr0 + 'top:' + (y1 + k*(x - x1)) + '; left:' + x + '">'
   x++
}
document.write(xstr)                 // запись в документ
}
```

Функция line() формирует строку, содержащую теги вывода экземпляров одного и того же графического изображения точки. При этом в цикле изменяются значения параметров top и left атрибута STYLE. Параметром цикла является горизонтальная координата, текущее значение которой присваивается параметру left. Таким образом, мы используем горизонтальную развертку линии. Вертикальная координата (top) вычисляется с учетом ее наклона — коэффициента, равного тангенсу угла наклона. Сформированная строка записывается в документ и отображается в окне браузера.

Функция дает неплохие результаты, когда линия проходит слева направо под углом к горизонтали, меньшим 45° (рис. 4.10). При больших углах становится заметной дискретность линии, что показано на рисунке. Это связано с тем, что количество отображаемых точек определяется величиной x2 - xl. Если мы хотим нарисовать вертикальную линию, то x2 = xl и, следовательно, количество отображаемых точек равно 0. Иначе говоря, для создания вертикальных линий рассматриваемая версия функции line() не приспособлена. Кроме того, эта функция рисует только линии, у которых начало находится левее и выше конца.

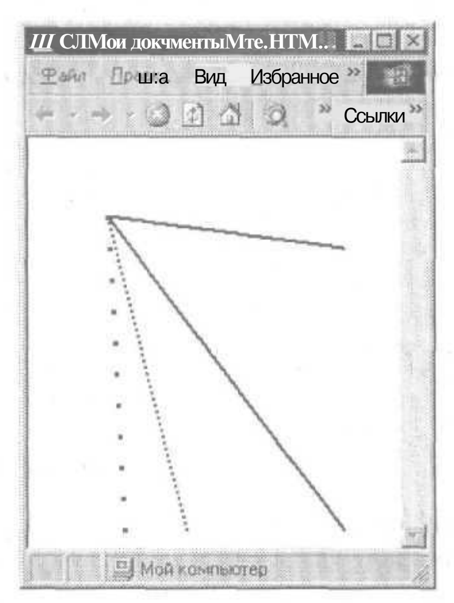


Рис. 4.10. Линии, нарисованные с помощью функции line() версии О

В следующей версии функции line() реализована простая идея, заключающаяся в том, чтобы выбирать подходящую развертку. Если линия ближе к горизонтали, чем к вертикали, то используется горизонтальная развертка, в противном случае выбирается вертикальная развертка (рис. 4.11). Кроме того, мы определяем направление развертки так, чтобы рисовать линии, у которых начало может быть как левее, так и правее конца (листинг 4.10).

Листинг 4.10. Код функции lineQ версии 1

```
function line(xl, yl, x2 ,y2, п){
/* Версия 1 */
/* xl, yl - начало линии
   x2, y2 - конец линии
   п - толщина линии
*/

var clinewidth = ""                    // строка, определяющая толщину линии
if (!n)
   clinewidth = ' WIDTH=" + n + " HEIGHT=" + n
var xstr=""                           // строка тегов для записи в HTML-документ
```

продолжение &

Листинг 4.10 *(продолжение)*

```
var xstrO = '<IMG SRC="point.bmp"' + clinewidtn + '
STYLE="position:absolute;'
var x, k, direct
var vertical = Math.abs(y2 - y1) > Math.abs(x2 - x1)
if (vertical)!
   direct = (y2 > yl)            // направление прямой
   x = yl
   к = (x2 - xl)/(y2 - yl)       // коэффициент наклона прямой
}else{
   direct = (x2 > xl)            // направление прямой
   if (direct) x = xl
   else x = x2
   k = (y2 - yl) / (x2 - xl)     // коэффициент наклона прямой
}

while (true) {
      if (!vertical){
      xstr += xstr0 + 'top: ' + (yl + k*(x - xl)) + '; left: ' + x+'">'
      if (x == x2) break
      if (direct) x*+
      else x--
   }else{
      xstr+=xstr0+ 'left: + (xl+k*(x-yl)) + '; top:' +x+ '">'
      if (x == y2) break
      if (direct) x++
      else x--
   }
}
document.write(xstr)             // запись в документ
}
```

Теперь не мешало бы получить возможность рисовать, например, штриховые линии, а также задавать их цвет. Чтобы задать цвет линии, необходимо выбрать под-

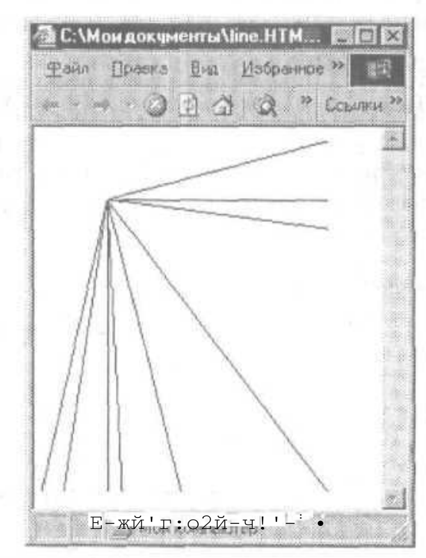


Рис. **4.11.** Линии, нарисованные с помощью функции lineQ версии 1

ходящий графический файл с изображением точки. Можно заранее заготовить файлы точек с различными цветами. Впрочем, можно заготовить файлы не только с изображениями точек, но и с произвольными изображениями. В конце концов, мы можем рисовать «линии», используя любые изображения в качестве точек. Для рисования штриховых линий необходимо просто периодически не отображать заданное количество точек. В следующей версии функции line() добавлены два параметра: имя графического файла и длина штриха линии (листинг 4.11). Предполагается, что длины штриха и паузы между соседними штрихами одинаковы. Если длина штриха не указана или равна 0, то линия сплошная.

Листинг 4.11. Код функции line() версии 2 (штриховые линии)

```
function line(pict_file, xl, yl, x2, y2, n, s){
/*  Версия 2 */
/* pict_file - имя графического файла
   xl, yl - начало линии
   x2, y2 - конец линии
   п - толщина линии
   s - длина штриха и паузы
*/
if (!pict_file)                        // файл с изображением точки
   pict_file = "point.bmp"
if (!s)                                // длина штриха и паузы
   s = 0
var clinewidth = ""                    // строка, определяющая толщину
                                       // линии
if (!n)                                // толщина
   clinewidth = " WIDTH=" + n + " HEIGHT=" + n
var xstr = ""     // строка тегов для записи в HTML-документ
var xstr0 = '<IMG SRC="' + pict_file + '"' + clinewidth + '
STYLE="position:absolute;"
var x, k, direct
var vertical = Math.abs(y2 - yl) > Math.abs(x2 - xl)
if (vertical){
   direct = (y2 > yl)                  // направление прямой
   x = yl
   k = (x2-xl) / (y2-yl)               // коэффициент наклона прямой
}else{
   direct = (x2 > xl)                  // направление прямой
   if (direct) x = xl
   else x = x2
   k = (y2 - yl)/(x2 - xl)             // коэффициент наклона прямой
I

var i = 0                              // счетчик точек штриха и паузы
var draw = true                        // рисовать или пропускать
while (true){
   if (!vertical){
   if (draw)
      xstr += xstr0 + 'top:' + (yl + k*(x - xl)) + '; left:' + x + '">'
      if (x == x2) break
      if (direct) x++
      else x--

   }else{
   if (draw)
```

продолжение •&

Листинг 4.11 *(продолжение)*

```
            xstr += xstr0 + 'left:' + (xl + k*(x - yl)) + '; top:' + x + '">'
            if (x == y2) break
            if (direct) x++
            else x--
        }
        if (s>0&&i>s){
            draw = Idraw
            i =O
        }
        i++
    }
    document.write(xstr)                    // запись в документ
    }
```

Вторая версия функции li"eQ позволяет не только рисовать сплошные и штриховые прямые линии, но и располагать вдоль них изображения из графических файлов (рис. 4.12). В последнем случае линии не видны, они выполняют лишь роль направляющих для расположения изображений. Подбирая изображения и длину штриха, можно получить интересные визуальные эффекты. Однако не следует забывать, что при больших размерах графического файла и значительном количестве «точек» процесс отображения на экране может занять много времени.

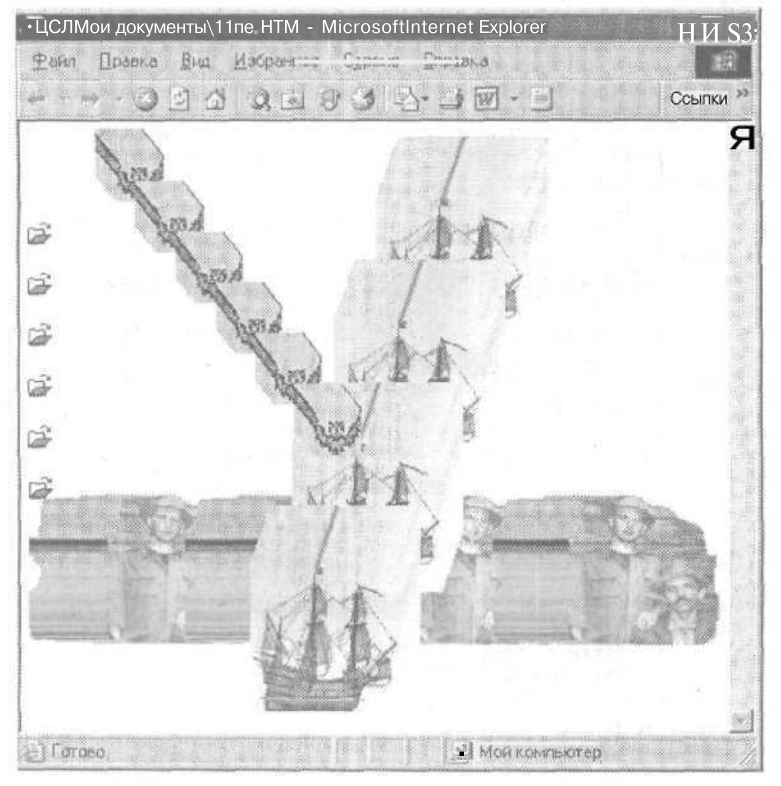


Рис. 4.12. Расположение изображений вдоль прямых штриховых линий

4.3.2. Произвольная кривая

А почему бы не создать функцию для рисования произвольных кривых, заданных некоторым выражением? На первый взгляд может показаться, что эта задача сложнее, чем рисование прямых линий. Но это не так. При написании кода функции мы используем большую часть опыта, полученного при решении задачи рисования прямых линий. Будем считать, что выражение, задающее кривую, содержит переменную, обозначенную строчной латинской буквой x. Тогда горизонтальная координата точки кривой вычисляется просто как значение переменной x, изменяемой в заданных пределах. Для вычисления вертикальной координаты используем функцию evalQ, которой передадим в качестве параметра строку, содержащую выражение (листинг 4.12).

Функция curve() для рисования кривой будет принимать следующие параметры: имя графического файла с изображением точки (впрочем, это может быть любое изображение), выражение, задающее кривую, координаты началалинии, количество точек линии, толщину линии и длину штриха (если потребуется штриховая линия) (рис. 4.13).

Листинг 4.12. Код функции curve()для рисования кривых

```
function curve(pict_file,yexpr, x0, y0, t, n. s){
/* Версия 0 */
/* pict_file - имя графического файла
   yexpr - выражение с переменной x
   x0, y0 - координаты начала кривой
   t - количество точек кривой (значений переменной x)
   п - толщина линии
   s - длина штриха и паузы
*/
if (lyexpr) return null
if ( !pict_file) pict_file = "point.bmp"
if (!s) s = 0
if (It) t = 0
var clinewidth = ''
if (!n)
   clinewidth = 'WIDTH=' + n + 'HEIGHT=' + n
var x
xstrG = '<IMGSRO"' "' + pict_file + '"' ' + clinewidth +
   ' STYLE = "position:absolute;top: '
xstr = ""
var i = 0, draw = true
for(x = 0; x < t; x++) {
   if (draw)
      xstr += xstrO + (y0 + eval(yexpr)) + '; left:' + (x0 + x) +'">'
   if (i > s&&s > 0) {
      draw = Idraw
      i = 0
   }
   i ++
}
document.write(xstr)                        // запись в документ
}
```

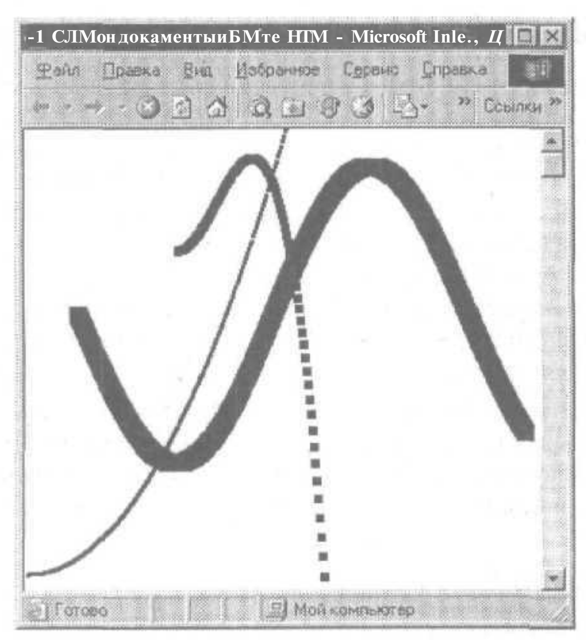


Рис. 4.13. Линии, нарисованные с помощью функции curveQ версии О

Ниже приведено несколько примеров:

```
curve("","200-0.01*x*x",1,100,200,3)                       // ветвь параболы
curve("","100.0*Math.sin(6/250*(x))",30,120,300,12,0)      // синусоида
curve("" , "0.001*x*x*(x-75)", "x" ,100,80,200,6,0)
```

Теперь модифицируем функцию curve() так, чтобы можно было рисовать замкнутые линии (например, окружность или эллипс). Для этого необходимо иметь возможность задавать выражение не только для вертикальной, но и горизонтальной координаты. Такой способ задания линии в математике называют параметрическим. В простейшем случае в качестве выражения для горизонтальной координаты можно задать как "x" (листинг 4.13). Тогда функция должна давать такой же результат, что и ее версия 0. Напомним, что в выражениях для координат точек линии переменная должна быть обозначена строчной латинской буквой x.

Листинг 4.13. Модифицированный код функции curveQ

```
function curve(pict_file, yexpr, xexpr, x0, y0, t, n, s){
/* Версия 1 */
/* pict_file - имя графического файла
   yexpr - выражение с переменной x для вертикальной координаты
   xexpr - выражение с переменной x для горизонтальной координаты
   x0, y0 - координаты начала кривой
   t - количество точек кривой (значений переменной x)
   n - толщина линии
   s - длина штриха и паузы
*/

if (!yexpr) return null
if (!xexpr) xexpr = "x"
if (!pict_file) pict_file = "point.bmp"
```

```
if (!s) s = 0
if (! t) t = 0
var clinewidth = ""
if (!n)
clinewidth = 'WIDTH=' + n + 'HEIGHT=' + n
var x
xstr0 = '<IMG SRC="' + pict_file + '"' + clinewidth +
   ' STYLE="positionrabsolute; top:'
xstr = ""
var i = 0, draw = true
for(x = 0; x < t; x++) {
   if (draw)
      xstr += xstr0 + (y0 + eval(yexpr)) + '; left:' + (x0 +
eval(xexpr)) + ' ">'
   if (i > s&&s > 0) {
      draw = Idraw
      i = 0
   }
   i++
}
document.write(xstr)             // запись в документ
}
```

Ниже приводятся несколько примеров так называемых фигур Лиссажу (рис. 4.14):

```
curve("", "80*Math.sin(6/25*x)", "60*Math. cos(6/25*x+l)" ,250, 100,300,2 ,0)
curve("" , "60*Math.sin(6/25*x)" , "60*Math . cos(6/25*x)" , 100 , 70 , 300,2 ,0)
curve("" , "80*Math.sin(6/25*x)" , "80* Math .cos (6/50*x)" ,100,200,600,6,0)
curve("", "80*Math.sin(6/250*x)", "80*Math. cos(6/75*x)",310,250 ,400,2 ,0)
```

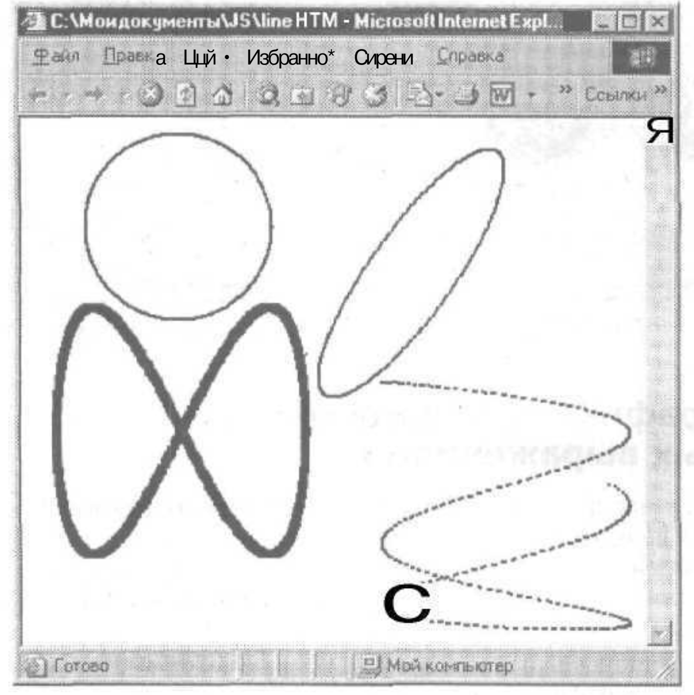


Рис. 4.14. Фигуры Лиссажу, нарисованные с помощью функции curveQ версии 1

Фигуры Лиссажу получаются, если выражения для координат представляют собой синус и косинус некоторого выражения с переменной x. Наиболее интересные кривые возникают, когда коэффициенты при переменной являются кратными друг другу.

Вместо изображения точки можно использовать любой другой рисунок. Подбирая графические файлы, вид кривой и длину штриха, можно получить интересные визуальные эффекты (рис. 4.15). Эти эффекты могут быть особенно впечатляющими, если использовать анимационные GIF-файлы. Однако помните, что при больших размерах графического файла и многочисленности рисунков процесс их отображения на экране может занять много времени.

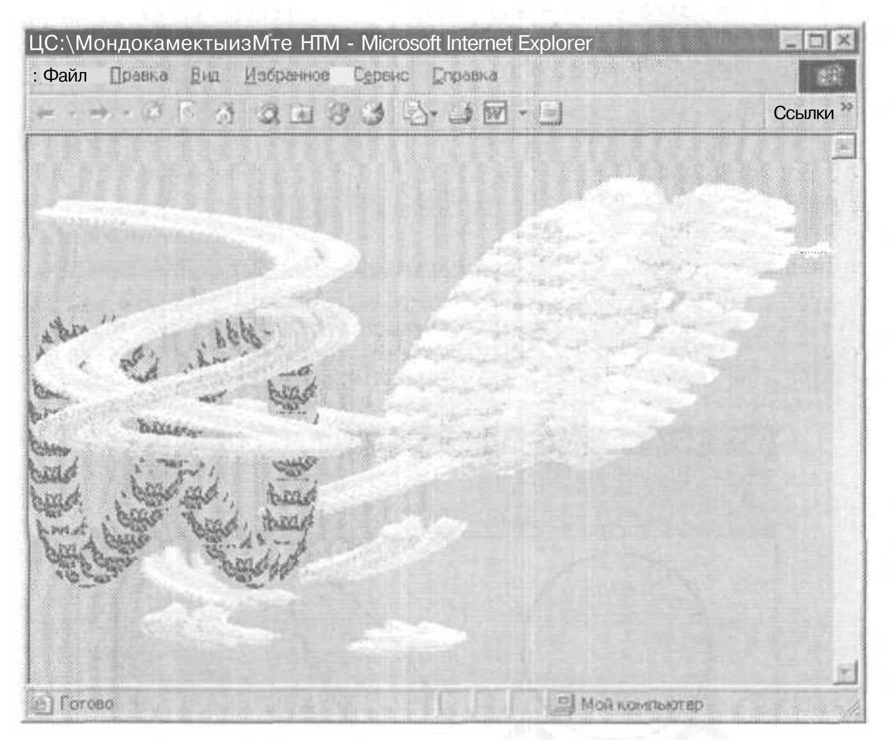


Рис. 4.15. Расположение рисунков вдоль фигур Лиссажу

4.3.3. Графики зависимостей, заданных выражениями

В принципе, у нас почти все готово, чтобы нарисовать график функции одной переменной в прямоугольной системе координат. Мы уже рассмотрели в предыдущих подразделах, как рисуются кривые, заданные выражением с одной переменной, а также прямые линии, необходимые для отображения осей координат. Теперь все это нужно собрать воедино.

Казалось бы, достаточно просто нарисовать две перпендикулярные прямые (оси координат), а затем кривую. Однако здесь возникают дополнительные задачи.

Во-первых, необходимо решить, как пересекаются координатные оси: какие квадранты они образуют и сколько. Во-вторых, требуется определить длину каждой из осей, которая, очевидно, зависит от максимального и минимального значений выражения, описывающего кривую. Наконец, следует задать способ оцифровки осей. Код программы, решающей эти задачи, получается значительно объемнее, чем код, обеспечивающий собственно рисование линий. Попробуйте написать его самостоятельно в качестве упражнения. В результате вы получите функцию, с помощью которой сможете рисовать графики различных зависимостей. Не забудьте при этом, что выше мы рассматривали отображение линий в экранной системе координат, в которой вертикальная ось направлена сверху вниз. В обычной практике рисования графиков вертикальную ось направляют, наоборот, снизу вверх.

4.3.4. Графики зависимостей, заданных массивами

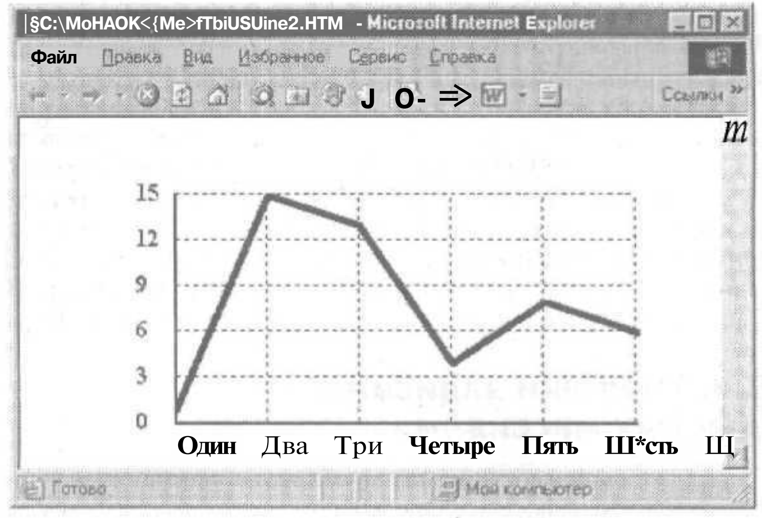
Как известно, зависимости между двумя величинами можно задавать в виде таблиц, состоящих из двух столбцов, в одном из которых располагаются данные, соответствующие аргументу зависимости, а в другом — ее значению. Аргументы отображаются на графике вдоль горизонтальной оси, а значения — вдоль вертикальной оси координат. Собственно зависимость обычно представляется либо точками на координатной плоскости, либо ломаной линией, проходящей через эти точки, либо вертикальными прямыми (столбиками). Возможны, конечно, и другие способы графического представления данных (например, в виде круговой диаграммы). Здесь мы остановимся на задаче построения графиков в прямоугольной системе координат в виде ломаной линии.

Таблицы с данными можно представить с помощью одного двухмерного или двух одномерных массивов. Будем считать, что имеем дело с двумя одномерными массивами, в одном из которых содержатся аргументы, а во втором — значения зависимости. Например, в одном массиве содержатся даты, а в другом — соответствующие этим датам среднесуточные (или какие-нибудь другие) значения температуры. Кривую зависимости между температурой и датой можно представить как ломаную линию, состоящую из отрезков прямых линий.

Легко догадаться, что для построения графика зависимости между элементами двух массивов в виде ломаной линии можно просто организовать в цикле вызов рассмотренной ранее функции **lineQ.** При этом следует заранее определить расстояние в пикселах между соседними точками по горизонтальной оси. Возникает задача масштабирования — перевода значений элементов массивов в пикселы экранных координат.

Прежде чем рассмотреть пример построения графика, немного модифицируем функцию Ипе(). Эта функция в том виде, как она была определена в подразделе «Прямая линия», не совсем подходит для наших целей. Сделаем так, чтобы она просто возвращала строку тегов, описывающую линию, а не записывала ее в документ. Для этого необходимо в определении функции line() заменить строку **document.**write(xstr) на return xstr.

В листинге 4.14 приводится код сценария, отображающего в окне браузера график зависимости, заданной с помощью двух массивов (рис. 4.16).



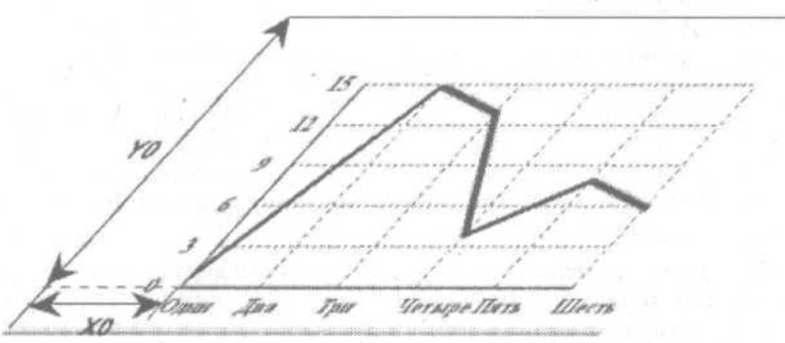


Рис. 4.16. График зависимости, заданной с помощью двух массивов

Листинг 4.14. Код сценария, отображающего в окне браузера график зависимости, заданной с помощью двух массивов

```
<SCRIPT>
/* График зависимости, заданной массивами */
/* Массив элементов горизонтальной оси */
var aX = new Array("Один", "Два", "Три", "Четыре", "Пять", "Шесть")
/* Массив элементов вертикальной оси */
var aY = new Array(1, 15, 13, 4, 8, 6)

var ky = 10, kx = 60                // коэффициенты масштабирования
var x0 = 100, y0 = 200              // отступы
var xstr = ""                       // строка тегов, описывающая линию

for(i = 0; i < aX.length -1; i++){  // линия графика
   x1 = x0 + kx*i
   y1 = y0 - ky*aY[i]
   x2 = x0 + kx*(i + 1)
   y2 = y0 - ky*aY[i + 1]
   xstr+= line("p_b.jpg", x1, y1, x2, y2, 4)
}
```

```
for (i=0; i < 6; i++){
/* Метки на вертикальной оси */
xstr+= "<b style='position:absolute;top:"+(y0-i*30-
10)+";left:75'>"+i*3+"</b>"
   if (i > 0){
                                          // горизонтальные линии:
     xstr+= line("point.bmp", x0, y0 - i*30, x0 + 5*kx, y0 - i*30, 1, 2)
                                          // вертикальные линии:
     xstr+= line("point.bmp ", x0 + kx*i , y0, x0 + kx*i, 50, 1, 2)
   }
/* Метки на горизонтальной оси */
xstr+= "<b style='position:absolute;top: " + (y0 + 5) + ";left:"+ (x0 +
  kx*i) +'">" +
aX[i]   + "</b>"
}
xstr+= line("point.bmp " ,x0,y0 ,x0 , 50, 2)          // вертикальная ось
xstr+= line("point.bmp ",x0,y0,x0 + 5*kx,y0,2)      // горизонтальная ось

document.write(xstr)                                 // запись в документ
</SCRIPT>
```

В этом сценарии масштабирование не универсально, а выбрано вручную на основе анализа конкретных исходных данных. Однако можно написать более универсальный код, автоматически формирующий масштабирующие коэффициенты на основе программного анализа исходных данных и (или) параметров, указанных пользователем. Попробуйте написать такой сценарий в качестве упражнения.

4.3.5. Динамические линии

Допустим, требуется время от времени перерисовывать линии. Например, если стрелка на изображении циферблата часов создается с помощью сценария на основе функции **Ипе()**, то для создания эффекта движения необходимо удалить ранее нарисованную стрелку и создать новую, положение которой соответствует текущему времени. В другом случае может потребоваться перерисовать график некой зависимости (например, курса валют от даты) при динамическом обновлении данных. Иначе говоря, в общем случае перерисовывание линии состоит из удаления ранее нарисованного ее изображения и создания новой линии.

Рассмотренные выше коды функций **lineQ** и **curve()** не вполне приспособлены к решению этой задачи из-за того, что они выполняют запись в документ сгенерированной строки HTML-тегов. Вообще говоря, это делать совсем не обязательно. Вполне достаточно, чтобы функции line() и curveQ просто возвращали строку тегов, а записать ее в HTML-документ можно и во внешней программе. В той же программе мы можем удалить и всю совокупность тегов, формирующих линию (когда это понадобится).

Итак, прежде всего изменим определения функций lineQ и curve() следующим образом: заменим в них выражение **document.write(xstr)** на return xstr. В результате функции не будут ничего рисовать, а будут лишь создавать и возвращать данные для последующего рисования. Теперь, чтобы нарисовать произвольную линию, следует в сценарии написать такую строку кода:

```
document.write(curve(параметры))
```

Очевидно, вместо одной этой строки можно записать следующие две:

```
var xstr = сигуе(параметры)
   document.write(xstr)
```

Поскольку мы должны быть готовы удалить линию из документа, строку тегов, возвращаемую функциями curveQ и line(), сначала заключим в какой-нибудь контейнер, например <DIV ID = . . .> с атрибутом ID, а затем запишем в документ:

```
var cmycurve="<DIV  ID = 'mycurve' > " i- сигуе(параметры) + "</DIV>"
document.write(cmycurve)    // запись в документ
```

Чтобы удалить линию из документа, достаточно **заменить** содержимое контейнерного тега <DIV> на пустую строку:

```
document.all.mycurve.innerHTML = ""
```

Здесь мы воспользовались свойством innerHTML, которое содержит весь HTML-код, заключенный в указанный контейнер. Присвоение этому свойству нового значения сразу же изменяет соответствующий элемент в документе. В данном случае вся последовательность тегов <IMG .. .>, формирующая линию, заменяется пустой строкой, в результате чего изображение линии исчезает. При этом сам контейнер <DIV> с пустым содержимым остается. В случае необходимости его можно наполнить новыми данными. Это типичный прием динамического изменения элементов HTML-документов.

Чтобы вместо предыдущей линии нарисовать новую, достаточно записать в сценарии следующий код:

```
document.all.mycurve.innerHTML = сигуе(другие_параметры)
```

В результате контейнер <DIV> заполнится последовательностью тегов <IMG. . .>, формирующих новую линию, а браузер ее отображает.

В качестве примера, позволяющего убедиться, что все это работает, я предлагаю следующий сценарий (листинг 4.15).

Листинг 4.15. Код динамических линий

```
<HTML>
<BUTTON onclick="redrawn">Перерисовать</B11TTOM>
<SCRIPT>
   /* Строка, возвращаемая curve() заключается в контейнер с заданным
      идентификатором ID */

   var cmycurve="<DIV ID = 'mycurve'>" + curve("", "80*Math.sin(6/25*x)",
      "80*Math.cos(6/50*x)",100,200,600,6,0) + "</DIV>"
   document.write(cmycurve)      // запись в документ и отображение линии

   /* Определения функций */
   function redrawO{          /* обработчик щелчка на кнопке
                                 (перерисовка линии) */
      document.all.mycurve.innerHTML = curve("", "60*Math.sin(6/25*x)",
        "60*Math.cos(6/25*x)" ,100,150,300,2,0)
   }

   function curve(pict_file, yexpr, xexpr, x0, y0, t, n, s){  // линия
   if  (lyexpr) return null
   if  (Ixexpr) xexpr = "x"
   if  (!pict_file)  pict_file  =  "point.bmp"
   if (!s) s = 0
   if (!t) t = 0
   var clinewidth = ""
```

```
    if (!n)
       clinewidth = 'WIDTH=' + n + 'HEIGHT=' + n
    var x
    xstr0 = '<IMG SRC="' + pict_fHe + '"' ' + clinewidth + ' STYLE =
       "position:absolute;top: '
       xstr = ""
    var i = 0, draw = true
    for(x =0; x < t; x++) {
    if (draw)
       xstr += xstrO + (y0 + eval(yexpr)) + '; left:' + (x0 + eval(xexpr))
         +'">'
    if (i >s && s > 0) {
       draw = ldraw
       i = Э
    }
    i++
    }
     return xstr                    // строка тегов, формирующих линию
     }
  </SCRIPT>
  </HTML>
```

При загрузке данного HTML-документа в окне браузера появляется кнопка и некоторая фигура Лиссажу. Щелчок на кнопке заменяет эту фигуру окружностью. Повторный щелчок уже не приводит к видимым изменениям, поскольку текущее изображение заменяется таким же.

4.4. Напишем число словами

При создании различного рода финансовых документов (например, платежных поручений) обычно возникает неоходимость представить числа прописью. Суть этой задачи заключается в том, чтобы преобразовать заданное число в строку, разбить ее на группы разрядов числа (единицы, десятки, сотни, тысячи и т. д.), проанализировать содержимое каждой группы и перевести его в некоторое слово. Специфика задачи заключается в выборе нужных окончаний этих слов, то есть в учете падежа, рода и числа имен существительных и числительных русского языка. Поясним это на следующем примере:

31 — тридцать один
31000 - тридцать одна тысяча
31000000 - тридцать один миллион

Многим программистам когда-то приходилось решать эту задачу. Однако в большинстве случаев, по моим наблюдениям, они ограничивались лишь именительным падежом и единственным числом, предоставляя пользователю самостоятельно скорректировать отображаемое число прописью с помощью клавиатуры. Так в свое время поступал и я, полагая, что учет окончаний имен числительных слишком сложное (или рутинное) дело. В конце концов я взял карандаш и бумагу и разобрал все возможные случаи. Оказалось, что их не так уж и много. В результате я написал код, который приводится в листинге 4.16.Я и сам удивился, что он оказался небольшим. Возможно, кое-что можно было бы сделать изящнее. Однако сама идея программы представляется мне вполне красивой.

Функция Nsay(N), решающая рассматриваемую задачу, принимает в качестве параметра N целое число и возвращает строку, которая представляет это число в виде последовательности слов. Эта функция использует ряд вспомогательных функций: speakQ, **LeftQ** и Right(). Если speakQ является специфической функцией, предназначенной для решения поставленной задачи, то функции LeftQ и **RightQ** довольно универсальны и могут быть использованы в других приложениях. Так, функция LeftQ возвращает левую часть указанной строки, а RightQ — правую ее часть. В принципе, можно обойтись и без этих функций, но с ними как-то удобнее. В представленном ниже варианте функция **NsayQ** позволяет представить прописью любое целое число, не превышающее 999 999 999 999 (один триллион минус единица).

Листинг 4.16. Код для представления числа словами

```
function Nsay(N) {                    // Перевод чисел в имена числительные
/* Параметр: целое число
Возвращает строку символов
• 7
if (N == null)
   return ""
if (!N)
   return "ноль"
if (Math.abs(N) > 999999999999)
   return "****"                     // превышение допустимого предела

var NN1 = "", NN2 = "". NN3 = "", NN4 = "", nNNl, nNN2, nNN3, nNN4, znak.
  xyz

/* Единицы */
ed = new Array(" один", " два", " три", " четыре", " пять", " шесть",
  " семь", " восемь", " девять")
/* Десятки */
de = new Array(" десять", " двадцать", " тридцать", " сорок", "
пятьдесят", " шестьдесят", " семьдесят", " восемьдесят", " девяносто")
/* Второй десяток */
dd = new Array(" одиннадцать", " двенадцать", " тринадцать",
  " четырнадцать", " пятнадцать", " шестнадцать", " семнадцать",
  " восемнадцать", "девятнадцать")
/* Сотни 7
so = new Array(" сто", " двести", " триста", " четыреста", " пятьсот",
  " шестьсот", " семьсот", " восемьсот", " девятьсот")

znak = "                              // знак числа
if (N < 0)    гпак="минус "
/* преобразуем число N в строку
   с ведущими нулями: */
NN = Right("000000000000" + Itrim(N.toStringO).12)
/* Выделяем группы разрядов, преобразуя их в числа.
Второй параметр функции parselntO, равный 10, здесь важен, поскольку
группы разрядов могут содержать ведущие нули, что может быть расценено
как 8-е число

nNNl = parse!nt(Left(NN,3), 10)
nNN2 = parse!nt(NN.substr(3,3), 10)
nNN3 = parse!nt(NN.substr(6,3), 10)
nNN4 = parse!nt(NN.substr(9,3), 10)
```

```
if  (nNNl >0 ) {
    NN1 =speak(nNNl)
    xyz = Right(NNl , 1)
    NN1 = NN1 + " миллиард"
    if (xyz == "a" || xyz == "и" || xyz == "e")
       NN1 = NN1 + "a"
    else <
       if(xyz != "H")
       NN1 = NN1 + "OB"
}
)

if  (nNN2 > 0){
    NN2 = speak(nNN2)
    xyz = Right(NN2,l)
    NN2 = NN2 + " миллион"
    if (xyz == "a" || xyz == "и" || xyz == "e")
       NN2 = NN2 + "a"
    else {
       if(xyz != "H")
          NN2 = NN2 + "OB"
}
}

if  (nNN3 > 0){
    ed[0] = " одна"
    ed[1] = "две"
    NN3 = speak(nNN3)
    xyz = Right(NN3,l)
    NN3 = NN3 + " тысяч"
    if (xyz == "и" || xyz == "e")
       NN3 = NN3 + "и"
    else {
       if  (xyz =="a")
          NN3 = NN3 + "a"
 I
ed[G] = " один"
ed[l] = "два"
}
if  (nNN4 > 0)
    NN4 = speak(nNN4)

NN4 = znak + NN1 + NN2 + NN3 + NN4
return !NN4 ? "ноль" : NN4

}

/* Вспомогательные функции : */
function speak (k) {
var yN1, yN2
if (k == 0)
   return ""
if (k == 10)
   return " десять"
if (k = 1 && k < 10)
   return ed(k-1]
if (k >= 11 && k < 20)
   return dd[k - 11]
```

продолжение

Листинг 4.16 *(продолжение)*

```
if (k >= 20 && k <= 99) {
   ск = ltrim(k.toStringO)
   yNl = parselnt(l_eft(ck, 1))
   yN2 = parselnt(ck.substr(l, 1))
   return yN2==0 ? de[yNl-l] : de[yNl - 1] + ed[yN2-l]
}
if (k >= 100 && k <= 999){
   к = ltrim(k.toStringO)
   return so[parseInt(Left(k,l)) -1] + speak(parselnt(k.substr(1, 2)))
   }
}
function Itrim(xstr){                // удаляет передние пробелы
 if (!(xstr.indexOf(" ") == 0))
   return xstr
var astr = xstr.split(" ")           // создаем массив из слов строки
var i = 0
while (i<astr.length){
   if (astr[i] != (""))
      break                          // выходим из цикла, если элемент не пуст
i++
}
astr = astr.slice(i)                 // создаем массив
return astr.join(" ")                // склеиваем элементы массива в строку
}

function Left(xstr, n){              // выделяет левую часть строки
return xstr.substr(0 , n)
}
function Right(xstr, n) {            // выделяет правую часть строки
return xstr.substr(xstr.length - n)
}
```

Ниже приведен HTML-код со сценарием, использующим функцию NsayQ. В окне браузера представляется поле ввода данных (рис. 4.17). Если ввести в это поле некоторое число и щелкнуть кнопкой мыши где-нибудь вне его, то это число отобразится в виде соответствующей последовательности слов.

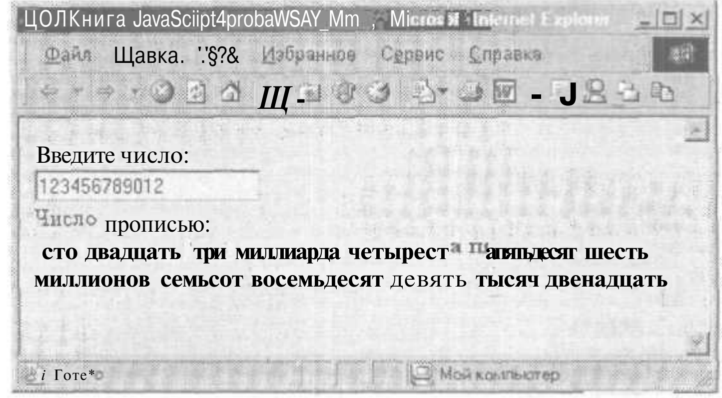


Рис. 4.17. Представление введенного числа прописью

```
<HTML>
<BODY bgcolo = "e0eOe0">
Введите  число:<BR>
<INPUT NAME = "myinput" TYPE = "number" VALUE = "" onchange =
  "numberchangeO">
<BR> Число прописью:<BR>
<B ID = "Ntext"></B>
</BODY>
<SCRIPT>
function numberchangeO{
document.all.Ntext.innerText=Nsay(document.all.my input.value)
}
</SCRIPT>
</HTML>
```

4.5. Обработка данных форм

Такие элементы HTML-документа, как поля ввода данных, текстовые области, переключатели и флажки, раскрывающиеся списки и кнопки, можно объединить в так называемую форму (рис. 4.18). В HTML форма создается с помощью контейнерного тега <FORM>, внутри которого располагаются теги элементов этой формы. В объектной модели документа каждой форме соответствует свой объект, входящий в коллекцию forms. Заметим, что любой из перечисленных выше элементов можно использовать вне всякой формы. Однако форма — не просто контейнер, а контейнер и объект, предназначенные главным образом для организации отправки на сервер всех данных, имеющихся в элементах этой формы (например, введенных пользователем). Давным-давно, когда браузеры воспринимали только простой HTML и не работали со сценариями, форма была единственным средством поддержки интерактивности.

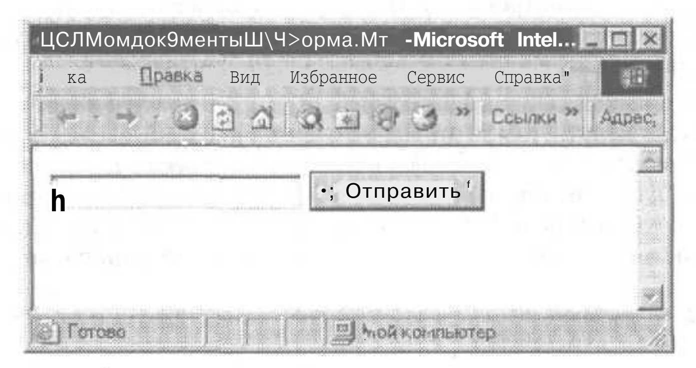


Рис. 4.18. Внешний вид формы в окне браузера

Как раньше, так и теперь для отправки данных на сервер сценарий не обязателен. Чтобы отправить данные, достаточно в теге <FORM> указать атрибут ACTION, а в самой форме установить кнопку типа Submit. Щелчок на этой кнопке инициализирует отправку данных. Если атрибут ACTION не указан или его значение пусто, данные формы не будут отправлены, даже если вы щелкнете на кнопке типа Submit.

Итак, для отправки данных формы атрибут ACTION должен иметь некоторое значение. В общем случае это URL-адрес файла или CGI-программы, которая получает и обрабатывает отправленные данные. Например, ACTIO N =" http://www. myserver/cgi/myprogram.pl".

Если вы хотите отправлять данные формы по электронной почте, то значением ACTION является строка вида:

mail to:адрес_e-mail

Можно также указать тему (subject) сообщения:

mailto:aflpec_e-mail.?subject=TeMa_coo6i«eHHfl

Кроме атрибута ACTION в теге <FORM> следует указать еще два атрибута: METHOD и ENCTYPE. Атрибут METHOD имеет значение POST или GET. Выбор значения отражается лишь на форме, в которой передаются данные. Если у вас нет особых причин задуматься об этом, выбирайте значение POST. Атрибуту ENCTYPE присвойте значение "text/plain". В этом случае отправляемое сообщение будет представлять собой последовательность пар вида имя_элемента=значение. Здесь имя_элемента — значение атрибута NAME в теге элемента, содержащегося в форме, а значение — значение атрибута VALUE в этом же теге. Если не указать атрибут ENCTYPE, то сообщение будет представлено в неудобочитаемом (закодированном) виде.

Вот пример HTML-документа с формой, содержащей поле ввода данных и кнопку типа Submit:

```
<HTML>
<FORM METHOD = POST ACTION = "mailto:mufliu@geras1in.ru"
  ENCTYPE="text/plain">
<INPUT NAME = "Сообщение" TYPE = "text" VALUE = "">
<INPUT NAME = "Отправить" TYPE="submit" VALUE = "Отправить">
</FORM>
</HTML>
```

Отправка данных рассмотренной выше формы произойдет при щелчке на кнопке типа Submit, на которой в нашем примере выводится надпись «Отправить». Адрес получателя указан как значение атрибута ACTION в теге <FORM>.

Если перед отправкой данных формы требуется предварительно их проверить или еще что-нибудь сделать, то для этого необходим сценарий. В следующем примере проверяется, имеется ли символ «@» в поле ввода адреса электронной почты получателя и не пусто ли поле ввода собственно сообщения. Если символа «@» в адресе нет или поле сообщения пусто, то отправка не производится. Сценарий обрабатывает событие onsubmit, возникающее при щелчке на кнопке типа Submit.

```
<HTML>
<FORM ID = "myform" METHOD=POST ACTI.ON="" ENCTYPE="text/plain"
style="background:'eЭeOeO'">
Кому:
<INPUT NAME = "email_to" TYPE = "text" VALUE = "">
<P>
От кого:
<INPUT NAME="email_from" TYPE = "text" VALUE = "">
<P>
Сообщение:<BR>
<TEXTAREA NAME = "Сообщение" TYPE = "text" VALUE = " "></TEXTAREA>
<P>
<! Кнопка типа Submit >
```

```
<INPUT NAME = "Отправить" TYPE = "submit" VALUE = "Отправить">
</FORM>
<SCRIPT>
function myform.onsubmit (){
var noemail = myform.email_to.value.indexOf('@') == - 1
                                                  /* присваиваем
                                                     значение
                                                     равенства */
var notext = Imyform.Сообщение.value
var xtext = "ХпПисьмо не отправлено"
if (noemail || notext){
   event.returnValue = false                      // отменить
отправку
   if (noemail)
      alert("Неправильный адрес получателя" + xtext)
   else
      alert("НеТ текста сообщения" + xtext)
} else
   myform.action = "mailto: " + myform.email_to.value // значение ACTION
)
</SCRIPT>
</HTML>
```

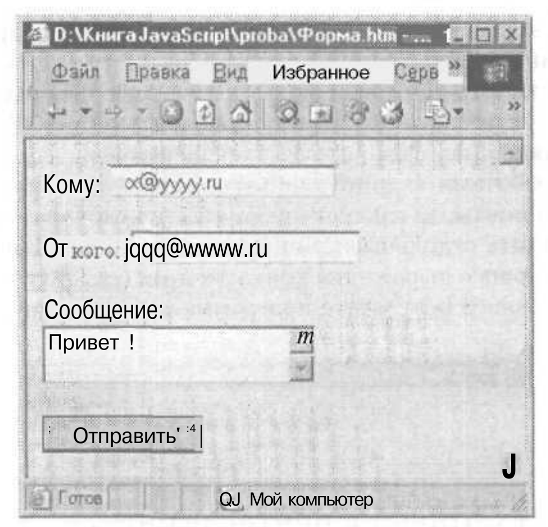


Рис. 4.19. Форма отправки сообщения по электронной почте

Если отправить данные формы, показанной на рис. 4.19, то адресат получит сообщение в следующем виде:

```
email_to=xxx@yyyy.ru
email_from=qqq@wwww.ru
Сообщение=Привет
Отправить=0тправить
```

В приведенном выше примере мы делаем поверхностную проверку данных, введенных пользователем в форму. Если данные не удовлетворяют нашему критерию правильности, то мы должны предотвратить их отправку. Это можно сделать двумя способами. Первый способ заключается в присвоении свойству returnValue

значения false (как в нашем примере). Тогда отменяется стандартная реакция на событие, в данном случае — реакция на событие onsubmit (отправка сообщения). Второй способ заключается в присвоении свойству action пустого значения. В нашем примере значение action пусто по умолчанию, поэтому выражение event.returnValue = false является излишним, но зато делает код сценария более понятным. В контексте нашего HTML-кода сценарий мог бы выглядеть и следующим образом:

```
<SCRIPT>
function myform.onsubmit() {
var noemail = myform.email_to.value.indexOf('@') == -1
var notext = !myform.Сообщение.value
var xtext = "ХпПисьмо не отправлено"
if (noemail || notext) {
   if (noemail)
      alert("Неправильный адрес получателя" + xtext)
   else
      alertC'HeT текста сообщения" + xtext)
} else
   myform.action = "mailto:" + myform.email_to.value // значение ACTION
}
</SCRIPT>
```

На веб-страницах нередко размещают ссылку или кнопку, открывающую форму отправки сообщения по электронной почте автору страницы (рис. 4.20). В этом случае не нужно вводить почтовый адрес получателя. Очевидно, что этот адрес уже присвоен в качестве значения атрибуту ACTION в теге <FORM>. Если вы хотите затруднить его выявление отправителем сообщения или, что важнее, различными программами-роботами, сканирующими веб-страницы с целью выявления адресов электронной почты, то следует предпринять некоторые меры. Самый простой рецепт — хранить отдельные компоненты адреса в различных переменных и собирать их с помощью выражения конкатенации (склейки), которое присваивается свойству action. В результате программа-робот не найдет в HTML-доку-

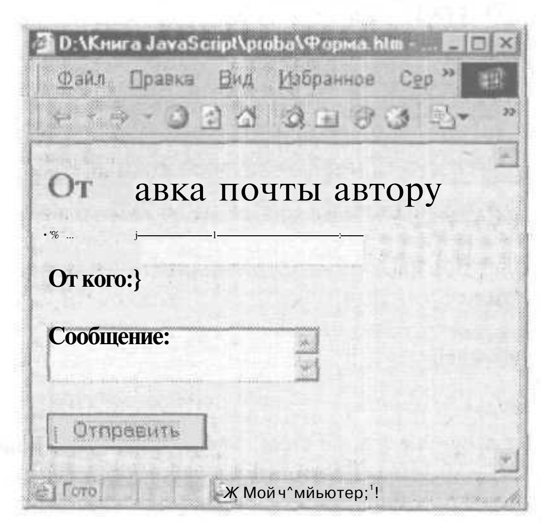


Рис. 4.20. Форма отправки сообщения по электронной почте автору веб-страницы

менте строки символов, имеющей структуру адреса электронной почты. Ниже приводится пример HTML-кода, воплощающий эту нехитрую идею.

```
<HTML>
<FORM ID = "myform" METHOD=POST ACTION=" " ENCTYPE="text/plain"
style="background:'e0e0e0'">
<H2>Отправка почты автору</H2>
От кого:
<INPUT NAME="email_from" TYPE = "text" VALUE = "">
<P>
Сообщение:<BR>
<TEXTAREA NAME = "Сообщение" TYPE = "text" VALUE = " "></TEXTAREA>
<P>
<! Кнопка типа Submit >
<INPUT NAME = "Отправить" TYPE = "submit" VALUE = "Отправить">
</FORM>
<SCRIPT>
var first = "mumu" , second = "gerasim"
function myform.onsubmit(){
if (Imyform.Сообщение.value){
   event.returnValue = false                  // отменить отправку
   alert("HeT текста сообщения ХпПисьмо не отправлено")
) else
   myform.action = "mailto:" + first + '"@" + second + ".ru"
                                              // значение ACTION
I
</SCRIPT>
</HTML>
```

Приведем еще один вариант привязки сценария обработки данных формы. Функционально он аналогичен рассмотренному выше.

```
<HTML>
<FORM ID = "myform" METHOD=POST ACTION=" " ENCTYPE="text/plain"
onsubmit = "return validatorO" " style ="background:'e0e0e0'">
<H2>Отправка почты автору</H2>
От кого:
<INPUT NAME="email_from" TYPE = "text" VALUE = "">.
<P>
Сообщение:<BR>
<TEXTAREA NAME = "Сообщение" TYPE = "text" VALUE = '" "></TEXTAREA>
<P>
<! Кнопка типа Submit >
<INPUT NAME = "Отправить" TYPE = "submit" VALUE = "Отправить">
</FORM>
<SCRIPT>
var first = "mumu", second = "gerasim"
function validatort(){
if (Imyform.Сообщение.value){
   alert("HeT текста сообщения ЛпПисьмо не отправлено")
   return false                              // отменить отправку
} else
   myform.action = "mailto: " + first + "'@" + second + ".ru"
                                             // значение ACTION
   return = true                             // разрешить отправку
}
</SCRIPT>
</HTML>
```

В этом примерефунцияvalidatorQ является обработчиком события **onsubmit** и возвращает либо true, либо false в зависимости от того, можно отправлять данные или нет. В теге <FORM> к событию onsubmit привязывается выражение return **val'dator();** что эквивалентно присвоению значения свойству **event.returnValue** в теле функции-обработчика события.

4.6. Меню

4.6.1. Раскрывающийся список

Простейшее меню можно создать с помощью тегов <SELECT> и <OPTION>. Обычно такие конструкции называют раскрывающимися списками. Ниже приводится простейший пример использования раскрывающегося списка (рис. 4.21). В этом примере раскрывающийся список задается HTML-кодом, а обработка выбора из этого списка — сценарием. Задача сценария заключается просто в обработке номера выбранного элемента из списка. В примере это вывод окна с соответствующим сообщением. Выбор пользователя из раскрывающегося списка производится щелчком левой кнопкой мыши на элементе списка. При этом свойство **selectedlndex** объекта элемента документа, соответствующего тегу <SELECT>, приобретает в качестве своего значения номер выбранного элемента списка (нумерация начинается с 0). Для инициации обработки выбора пользователя здесь служит событие onchange (произошло изменение в выделении элемента списка). Обработка этого события осуществляется функцией **myselectionQ.** Начальное выделение и отображение элемента в раскрывающемся списке задается атрибутом SELECTED тега <OPTION>. В рассматриваемом примере в списке виден элемент «Физика».

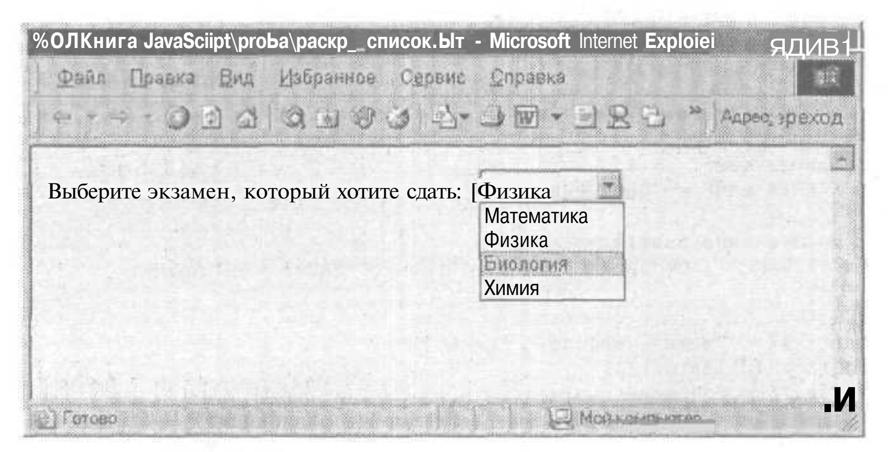


Рис. 4.21. Раскрывающийся список

```
<HTML>
Выберите экзамен, который хотите сдать:
<5ELECT NAME="TEST" onchange ="myselection()">
    <OPTЮM>Математика
    <OPTION SELECTED>4>M3MKa
    <OPTЮM>Биология
```

```
    <OPTION>Химия
</SELECT>
<SCRIPT>
function myselection() {
var testname, testnumber;
testnumber = document.all.TEST.selectedIndex
if (testnumber == 0)
    testname="Мат. анализ"
else{
    if (testnumber == 1)
        testname="Квантовая физика"
    else{
        if (testnumber == 2)
            testname="Биология"
        else
            testname = "Органическая химия"
    }
}
alert("Вы будете сдавать: " + testname)
}
</SCRIPT>
</HTML>
```

По умолчанию в раскрывающемся списке виден только один элемент. Чтобы раскрывающийся список сразу был приоткрытым (показывал несколько элементов), следует в теге <SELECT> указать атрибут SIZE = **количество_видимых_элементов** (рис. 4.22).

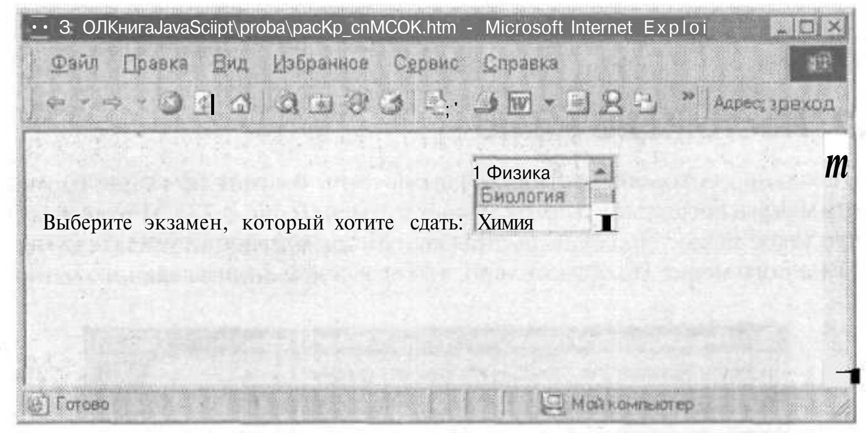


Рис. 4.22. Раскрывающийся список, в котором видны сразу три элемента

Пользуясь регулярностью HTML-кода, задающего раскрывающийся список, нетрудно сгенерировать его с помощью сценария (листинг 4.17).

Листинг 4.17. Код сценария раскрывающегося списка

```
<HTML>
Выберите экзамен, который хотите сдать:
<SCRIPT>
var N_sel = 1              // номер элемента, выбранного по умолчанию
var aoptions= new Array()  // массив элементов списка
aoptions[0] = "Математика"
aoptions[1] = "Физика"
```

продолжение ⇨

Листинг 4.17 *(продолжение)*

```
aoptions[2] = "Биология"
aoptions[3] = "Химия"
/* Строка, содержащая теги, формирующие раскрывающийся список */
xstr='<SELECT ID ="TEST" onchange = "myselection()">'
for (i = 0; i <aoptions.length; i++){
xprefix = (i == N_sel)? 'SELECTED=' +N_sel: ""
xstr+= '<OPTION ' + xprefix + '>' + aoptions[i]
}
xstr+= '</SELECT>'
document.write(xstr)           // запись в документ
function myselection() {
var testname, testnumber;
testnumber = document.all.TEST.selectedIndex
if (testnumber == 0)
   testname="Мат. анализ"
else{
   if (testnumber == 1)
      testname = "Квантовая физика"
   else{
      if (testnumber == 2)
         testname = "Биология"
      else
         testname = "Органическая химия"
   }
}
alert("Вы будете сдавать: "+ testname)
}
</SCRIPT>
</HTML>
```

4.6.2. Настоящее меню

Меню, создание которого мы сейчас рассмотрим, состоит из главного горизонтального меню и нескольких вертикальных подменю (рис. 4.23). Мне ли вам объяснять, что такое меню? Подменю раскрываются при наведении указателя мыши на опции главного мелю. Не обязательно, чтобы каждой опции главного меню соот-

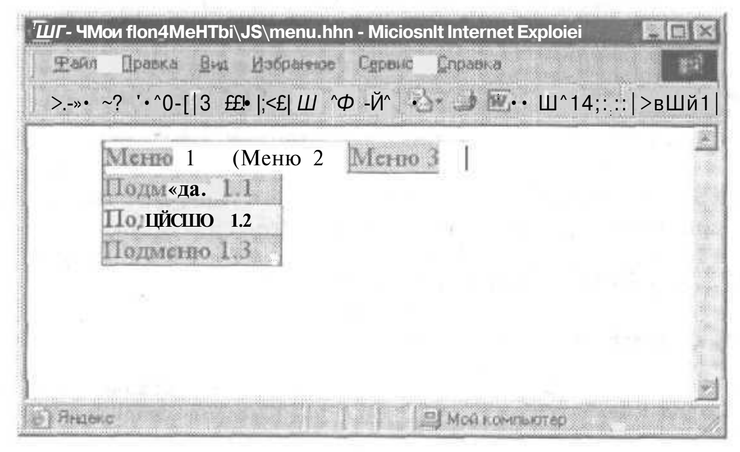


Рис. **4.23.** Внешний вид меню

ветствовало подменю: некоторые опции главного меню могут быть терминальными. При выборе опции щелчком левой кнопкой мыши происходит некоторое действие, определяемое сценарием. Это может быть URL-адрес веб-страницы или строка, содержащая код JavaScript.

Код сценария, реализующего такое меню, мы сделаем возможно более универсальным и оформим в виде двух файлов с расширением js. Первый файл, menu_prm.js, будет содержать параметры меню (листинг 4.18). При создании приложений его содержимое можно изменять в зависимости от конкретных задач. Например, в этом файле определяются конкретные названия опций и действия для них, цвет и другие параметры. Таким образом, это — переменная (настраиваемая) часть описания меню. Второй файл, menu_bld.js, будет содержать описание механизма построения и функционирования меню (листинг 4.19). В нем используются параметры, определенные в первом файле. Это постоянная часть описания меню. Разумеется, вы можете скорректировать содержимое этого файла по своему усмотрению.

Чтобы создать меню, в HTML-документе следует просто записать следующие строки:

```
<SCRIPT SRC = "menu_prm.js"x/SCRIPT>
<SCRIPT SRC = "menu_bld.js"x/SCRIPT >
<SCRIPT >buildMenu()</SCRIPT >
```

Здесь buildMenu() — функция, определение которой дано в файле menu_bld.js. Она выводит на экран меню в соответствии с параметрами, заданными в файле menu_prm.js.

Рассмотрим пример, в котором главное (горизонтальное) меню содержит три опции, первым двум из которых соответствуют вертикальные подменю. Последняя опция является терминальной. Названия опций выбраны так, чтобы было легко понять, что к чему относится.

В первом файле мы задаем параметры цвета, шрифта, а также состав названий опций и действий и координаты. Названия опций меню, их позиционирование, действия и содержание статусной строки задаются с помощью массивов. В конце концов, структура меню задается двухмерным массивом. Обратите внимание, что если действие представляет собой не URL-адрес, а некоторый код на языке JavaScript, то он должен начинаться с префикса "javascript:", за которым следуют выражения, разделенные точкой с запятой.

Во втором файле определен ряд функций, с помощью которых меню развертывается и функционирует. Отображение меню происходит на основе использования HTML-тегов, определяющих таблицы. Фрагменты HTML-кода, из которых складываются генерируемые строки HTML-документа, определены в виде элементов массива. Позиционирование меню производится с помощью параметров top и left таблицы стилей, которая также генерируется сценарием. Среди множества функций, определенных в этом файле, имеется главная — buildMenu(). Вызов ее в сценарии, расположенном в HTML-документе, выводит меню в окно браузера вместе с другими элементами этого документа.

Листинг 4.18. Код файла menu_prm.js

```
/* Параметры меню */
// Цвета:
var clBorder = 'blue'          // цвет рамки
```

продолжение &

Листинг 4.18 *(продолжение)*

```
var clBgInact = "#83d6f5"    // цвет фона
var clBgAct = '#d6f4fe'      // цвет подсветки фона опции
var clFnInact = 'blue'       // цвет надписи обычной
var clFnAct = 'black'        // цвет надписи при подсветке опции

// Шрифт (параметры стиля):
var cFontSize = '18'         // размер шрифта
var cFontFamily = 'times'    /* семейство шрифтов ('arial',
                                'courire', 'times' и т. п.) */

// Другие параметры:
var closeTimeout = 500       // время задержки свертывания подменю, мс
var selfPos = false          /* автоматическое позиционирование опций
                                меню и подменю (если true,
                                то задаются координаты только
                                первого меню) */

// Собственно меню:
var menu = new ArrayO
menu[0] = new ArrayO
/* Параметры главного меню:
   название опции;
   URL-адрес или строка с кодом JavaScript, начинающаяся с
 "javascript:";
   статусная строка;
   left (x-координата главного меню);
   top (y-координата главного меню) ;
   width (ширина главного меню);
   left (x-координата подменю)
   top (y-координата подменю);
   width (ширина подменю).
*/
menu[0][0] = new ArrayC'Меню 1","",,"Выберите что-нибудь из подменю", 50,
  10, 80, 50,120, 120)
menu[0][l] = new Array("Подменю 1.1" , "javascript:history.go(-1)",
  "Назад")
menu[0] [2] = new Array("Подменю 1.2" , "http://www.yandex.ru", "Яндекс")
menu[0][3] = new Array("Подменю 1.3" , "javascript:
  alert('Привет!')","Окно с приветствием")
menu[1] = new ArrayO
menu[1][0] = new ArrayC'Меню 2,"","", 130, 10, 80, 130, 120, 0)
menu[1][1] = new Array ("Подменю 2.1 ',"")
menu[1][2] = new Array("Подменю 2.2","","")
menu[2] = new ArrayO
menu[2][0] = new ArrayC'Меню 3", http://www.chat.ru", "CHAT.RU", 210,
  10, 80, 0, 0, 0)
```

Листинг 4.19. Код файла menu_bld.js

```
var ie = document.all ? true : false // есть ли что-нибудь в документе?
var overBox = ''
var timerID

// Заготовки элементов HTML-кода
var barHtml = new ArrayO
barHtml[0] = '<DIV ID="divbarpos'
barHtml[1] = '" STYLE="position:absolute;left:'
barHtml[2] = 'px; top: '
barHtml[3] = 'px;" onmouseover="openbox('
```

```
barHtml[4] = ' ) " onmouseout="closebox ('
barHtml[5] = ' ) " onclick="clickbox('
barHtml[6] = ' )"><TABLE CELLPADDING=0 CELLSPACING=0><TR><TD BGCOLOR="'
barHtml[7] = '"><TABLE CELLPADDING="0" CELLSPACING="1" BORDER = "0">
<TR><TD CLASS = "mnubarpos" ID = "mnubarpos'
barHtml[8] = "' WIDTH="'
barHtml[9] = '" BGCOLOR="'
barHtml[10] = "' STYLE="color:'
barHtml[11] = ' ;font-size:'+cFontSize+';font-family:'+cFontFamily+';">'
barHtml[12] = '</TD></TR></TABLE></TD></TR></TABLE></DIV>'
van boxHtml = new ArrayO
boxHtml[0] = '<DIV ID="divbox'
boxHtml[1] = '" STYLE="position:absolute;visibility:hidden;left: '
boxHtml[2] = 'px;top:'
boxHtml[3] = 'px ; font-family:'+cFontFami ly+'" onmouseout="closebox ('
boxHtml[4] = ') "><TABLE CELLPADDING=0 CELLSPACING=0><TR><TD BGCOLOR = "'
boxHtml[5] = '"xTABLECLASS="mnubox" ID="mnubox'
boxHtml[6] = ' " CELLPADDING="0" CELLSPACING="1" BORDER="0" >'
boxHtml[7] = '<SPAN ID="divboxpos'
boxHtml[8] = '" onmouseover="openboxpos('
boxHtml[9] = ' ) " onmouseout="closeboxpos('
boxHtml[10] = ') " onclick="clickboxpos('
boxHtml[11] = ') "><TR><TD CLASS = "mnuboxpos" ID="mnuboxpos'
boxHtml[12] = '" WIDTH="'
boxHtml[13] = ' " BGCOLOR="'
boxHtml[14] = '" STYLE="color:'
boxHtml[15] = ';font-size:'+cFontSize+'"> '
boxHtml[16] = '</TD></TRx/SPAN>'
boxHtml[17] = '</TABLE></TD></TR></TABLE></DIV>'

function buildMenuO {            // построение меню (главная функция)
   if (ie) {
      buildMenuBarO;
      buildSubMenuO ;
      if (selfPos) PosMenuO;
      ResizeSubMenuO ;
   }
}

function buildMenuBarO {         // построение горизонтального меню
   for (i =0; i < menu. length; i++){
var s = barHtml[0] + i + barHtml[1] + menu[i][0][3] + barHtml[2] +
  menu[i] [0] [4] +• bar Html [3] + i + bar Html [4] + i + barHtml [5] + i +
  barHtml[6] + clBorder +barHtml[7] + i + barHtml[8] + menu[i ] [0][5] +
  barHtml[9] + clBglnact + barHtml[10] + clFnlnact + barHtml[11] +
  menufi][0] [0] + barHtml[12];
      document.writeln(s);
   }
}

function buildSubMenuO {         // построение подменю
   for (i =0; i < menu.length; i++){
      if (menu[1].length > 1) {
var s = boxHtml[0] + i + boxHtmltl] + menu[i][0][6] + boxHtml[2] +
  menu[i][0][7] + boxHtml[3] + i + boxHtml[4] + clBorder + boxHtml[5] +
  i + boxHtml[6]
         for (j =1; ]< menu [i ]. length; j++) {
            var si = i + ' ' + i
            var s2 = i + ' j + J
```

продолжение ⇨

Листинг 4.19 *(продолжение)*

```
s += boxHtml[7] + s2 + boxHtml[8] + si + boxHtml[9] + si +
  boxHtml [10] + si + boxHtml [11] + s2 + boxHtml [12] .•menu[i] [9] [8] +
  boxHtml[13] + clBglnact + boxHtml[14] + clFnlnact • boxHtml [15] +
  menu[i][j][0] + boxHtml[16]
          }
          s += boxHtml[17]
          document.wri teln(s)
       }
    }
 i

 function PosMenu0 {
    for (i = 0; i < menu.length; i++) {
       barCurr = document.all['divbarpos' + 1]
       if (i > 0) {
          barPrev = document.all['divbarpos' + (i - 1)]
          barCurr.style.left = barPrev .offsetLeft +
            barPrev.offsetWidth - 1
       \
       if (menu [i ] . length > 1) {
          boxCurr = document.all['divbox' + i]
          boxCurr.style.pixelTop = barCurr.offsetTop +
            barCurr .offsetHeight - 1
          boxCurr. style.pixelLeft = barCurr .offsetLeft
       }
    }
 }

 function ResizeSubMenu() {
    for (i =0; i < menu.length; i++) {
       if (menu[i] .length > 1 && menu [i ] [0] [8] <= 0) {
          el = document.all['mnubox' + i]
          w = document.all['divbarpos' + i].offsetWidth
          if (el.offsetWidth < w) el.style.width = w
       }
    ]
 \

 function openbox(x) {
    paintLayer('mnubarpos' + x, active = true)
    showLayer('divbox ' + x)
    for (i =0; i < menu.length; i++) if (i != x) closebox(i, 0)
    overBox = 'divbox' + x
    window.status = menu[x][0][2]
 )

 function closebox(x, timeout) {
    paintLayer ('mnubarpos ' + x, active = false)
    clearTimeout (timerID)
    if (timeout == 0) hideLayer('divbox' + x)
    else timerlD = setTimeout ('hideLayer ("divbox' + x + '") ' ,
      closeTimeout)
    overBox = ' '
    window. status = defaultStatus
 }

 function clickbox(x) {
    clickMenu(menu[x][0][1])
 }
```

```
function openboxpos(x , y) {
   paintLayer('mnuboxpos' + x + '_' + y, active = true)
   overBox = 'divbox' + x
   window.status = menu[x][y][2]
}

function closeboxpos(x, y) {
   window.status = defaultStatus
   paintLayer( 'mnuboxpos' + x + '_' + y, active = false)
)

function clickboxpos(x, y) {
   clickMenu(menu[x][y][1])
}

function clickMenu(s) {
   if (s . indexOf (' javascript: ') == 0) eval(s);
   else if (s != '') window.location.href = s;
>

function paintLayer(layerID , active) {
   if (layer = document.all[layerlD]) {
      if (active) { clBg = clBgAct; clFn = clFnAct }
      else { clBg = clBglnact; clFn = clFnlnact }
      layer.style.backgroundColor = clBg
      layer . style.color = clFn
   }
}

function showLayer(layerID) {
   if (layer = document.all[layerID]) layer.style.visibility = 'visible'
}

function hideLayer(layerID) {
   if (layer = document. all[layerID] && (overBox != layerID))
     document.all[layerID].style.visibility = 'hidden'
}
```

Обратите внимание, что позиционирование элементов меню и подменю может происходить независимо друг от друга. Это позволяет визуально представить меню на экране в виде пространственно разнесенных частей (рис. 4.24). Таким образом, чтобы сделать несколько меню, не обязательно для каждого из них создавать и загружать специальные файлы с описаниями.

4.7. Поиск в текстовой области

Тексты большого объема, расположенные на веб-странице, обычно снабжают поисковой системой. Внешне она обычно выглядит как поле ввода поискового образа (набора символов, который требуется найти в тексте) и кнопки, щелчок на которой запускает процедуру поиска. В простейшем варианте эта процедура прокручивает текст в окне так, чтобы найденный поисковый образ оказался видимым и выделенным. Если поиск оказался неудачным, то положение текста в окне остается неизменным и, возможно, появляется соответствующее сообщение.

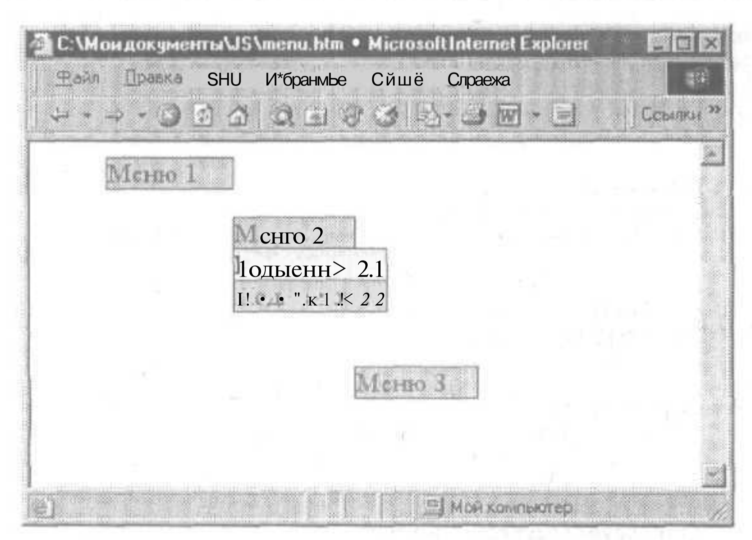


Рис. 4.24. Меню можно представить в виде пространственно разнесенных частей

Для решения задачи поиска в тексте используется объект TextRange. Этот объект просто обеспечивает доступ к текстовой информации, находящейся в объектах, которые соответствуют тегам <BODY>, <TEXTAREA>, <BUTTON> и <INPUT TYPE = **"text"**>.

В рассматриваемом ниже примере организуется поиск в тексте, находящемся в теле HTML-документа, то есть в контейнерном теге <BODY>. В этот же контейнер мы помещаем поле ввода поискового образа и кнопку для инициации поиска. Сценарий содержит описание функции myfind(), вызываемой при щелчке на кнопке. Функция myfind() с помощью метода createTextRange() создает текстовую область. Это не копия текста, содержащегося в теле документа, а просто ссылка на него. Далее с помощью метода findText() производится поиск в текстовой области строки символов, которую ввел пользователь в поле с именем WORD. Наконец, с помощью метода scrollIntoView() содержимое окна прокручивается так, чтобы найденная строка символов оказалась видимой. С помощью метода select() найденная строка в тексте выделяется цветом (подсвечивается).

```
<HTML>
<BODY>
<INPUT TYPE = "text" NAME = "WORD" VALUE = "" SIZE = 20 >
<BUTTON onclick = "myfind()">Поиск</BUTTON>
<P>

<! Здесь расположен текст, в котором производится поиск >

</BODY>
<SCRIPT>
function myfind() {
obj = document.body.createTextRange()   // создаем текстовую область
obj.findText(WORD.value)                // производим поиск
obj.scrollIntoView()                    /* прокручиваем текстовую
                                           область в окне */
obj.select()                            // выделяем найденное
}
```

```
</SCRIPT>
</HTML>
```

Если поиск неудачен (поисковый образ, введенный в поле ввода, не найден), то ничего не произойдет. А что будет, если пользователь нажал кнопку Поиск, когда ничего не было введено? Появится сообщение об ошибке. Чтобы исключить подобную неприятность, потребуется несколько изменить код программы. А именно следует проверить, не является ли введенный поисковый образ пустым. Ниже приведен вариант кода функции:

```
function myfind(){
if (IWORD.value) return         // если поисковый образ пуст, выходим
obj = document. body .createTextRangeO
obj .findText (WORD. value)
obj.scroll I ntoViewO
obj .select()
}
```

Теперь рассмотрим вариант, при котором на странице помимо текста присутствует только кнопка Поиск. Щелчок на этой кнопке должен привести к появлению формы, в которую пользователь должен ввести поисковый образ. Если этот поисковый образ окажется непустым, то запускается процедура поиска, в противном случае ничего не произойдет. В теле функции поиска использован стандартный метод **promptQ,** который принимает в качестве параметров пояснительный текст и начальное значение поискового образа (в данном случае — пустую строку) и предоставляет поле для ввода значения. В окне есть две кнопки — OK и Отмена. Метод promptQ возвращает введенное пользователем значение либо false, если пользователь щелкнул на кнопке Отмена. Ниже приводится описание функции:

```
function myfind(){
WORD = prompt("Введите, что найти: ","")
if (IWORD) return               // если поисковый образ пуст, выходим
obj =document. body. c• eateTextRangeO
obj .findText(WORD)
obj .scrollIntoViewO
obj .select ()
1
```

В качестве упражнения измените рассмотренные выше функции поиска таким образом, чтобы в случае неудачного поиска в тексте на экране появилось сообщение «Не удалось найти». Для вывода сообщения воспользуйтесь стандартным методом **alert().**

Ниже приводится пример организации поиска не в теле документа, а в текстовой области, заданной тегом <TEXTAREA>. Напомним, что внешне элемент <TEXTAREA> выглядит как прокручиваемое поле ввода. Если информация о теле документа (о содержимом контейнерного тега <BODY>) хранится в объекте body, то в случае элементов <TEXTAREA>, <BUTTON> и <INNPUT> необходимо обращаться к ним по значению атрибута ID или NAME.

```
<HTML>
<INPUT TYPE = "text" NAME = "WORD" VALUE = "" SIZE = 20 >
<BUTTON onclick = "myfind()">Поиск</BиTTOM>
<P>
<TEXTAREA ID = "mytext">
<! Здесь расположен текст, в котором производится поиск >
```

```
</TEXTAREA>
<SCRIPT>
function myfind() {
if (IWORD.value) return                     /* если поисковый образ
                                               пуст, выходим */
obj = document.all.mytext.createTextRangeO  /* создаем текстовую
                                               область */
obj.findText(WORD.value)                    // производим поиск
obj.scrollIntoViewO                         /* прокручиваем текстовую
                                               область в окне */
obj.select()                                // выделяем найденное i
1
</SCRIPT>
</HTML>
```

4.8. Таблицы и простые базы данных

Таблицы являются, пожалуй, наиболее часто используемыми элементами в веб-дизайне. Это обусловлено простотой и удобством компоновки документа с помощью тегов таблицы, таких как <TABLE>, <TR>, <TD> и др. Вы разбиваете все пространство окна на прямоугольные ячейки с видимыми или невидимыми границами и размещаете в них элементы документа (изображения, тексты, ссылки, кнопки, другие таблицы и т. п.). Таким образом, таблицы выполняют роль каркаса документа. До появления CSS они оставались единственным средством точного позиционирования элементов, но и сейчас широко используются как мощный инструмент для понятной организации и эффектного представления информации.

4.8.1. Доступ к элементам таблицы

Таблица как объект документа имеет две коллекции, посредством которых осуществляется доступ к ее содержимому. Первая из них — коллекция строк rows, а вторая — коллекция ячеек **cells.** Коллекция rows содержит все строки таблицы, включая разделы, соответствующие тегам <THEAD> и <TFOOT>. Коллекция cells содержит все элементы таблицы, созданные с помощью тегов < T H> и <TD>. Доступ к элементам коллекций осуществляется либо по индексу, либо по значению атрибута ID в соответствующем теге. Так, для доступа к строке таблицы можно использовать значение ID в теге <TR>, а для доступа к ячейке — значение ID в теге -TH> или <TD>. При использовании индекса (номера) следует иметь в виду, что нумерация начинается с 0. При этом ячейки таблицы нумеруются слева направо и сверху вниз.

Рассмотрим пример простой таблицы, содержащей три столбца и четыре строки (рис. 4.25):

```
<HTML>
<TABLE ID = "mytab">
<THEAO>Моя таблица</THEAO>
<TH>Фамилия</TH><TH>Имя</TH><TH>Должность</TH>
<TR ID="rl">
<TD>HBaHOB</TD><TD>MBaH</TD><TD>£HpeKTop</TD>
</TR>
<TR ID="r2">
```

```
<TO>Петров</TO><TO>Петр</TO><TO>Заместитель директора</TD>
</TR>
<TR ID="r3">
<TD>CMfloроВа</TD><TD>3nno4Ka</TD><TD>CeKpeTapma</TD>
</TR>
<TR ID="r4">
<TO>Федоров</TO><TO>Федор</TO><TO>Водитель</TO>
</TR>
</TABLE>
</HTML>
```

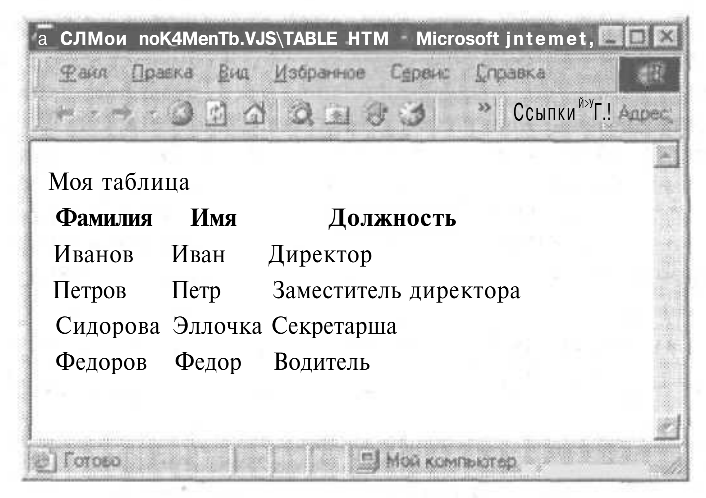


Рис. 4.25. Пример таблицы

Ниже приведено несколько примеров ссылок на элементы таблицы из сценария:

```
document.all.mytab.rows[0]      // ссылка на строку заголовков столбцов
document.all.mytab.rows[1]      // ссылка на первую строку данных
document.all.mytab.rows["rl" ]  /* другой способ ссылки на первую
                                   строку данных */

document.all.mytab.cells[0]     /* ссылка на ячейку, содержащую
                                   "Фамилия" */
document.all.mytab.cells[3]     // ссылка на ячейку, содержащую "Иванов"
document.all.mytab.cells[4]     // ссылка на ячейку, содержащую "Иван"
document.all.mytab.cells[6]     // ссылка на ячейку, содержащую "Петров"
```

Чтобы обратиться к строке или ячейке таблицз, следует учесть иерархию объектов: сначала мы обращаемся к коллекции all всех элементов документа, затем к таблице и только после этого — к элементу таблицы. Обратите внимание, что эти ссылочные выражения возвращают не содержимое элементов таблицы (строк или ячеек), а лишь ссылки на элементы как на объекты.

Каждая строка таблицы (элемент коллекции rows), являясь объектом, имеет свою собственную коллекцию ячеек, которая называется cells, так же как и коллекция всех ячеек таблицы. Однако нумерация ячеек в этой коллекции происходит в пределах одной строки начиная с 0. Например, в случае трехстолбцовой таблицы ячейки из коллекции cells для одной строки имеют индексы 0,1и *2.*

Примеры

```
document.all.mytab.rows[0].cells[0]       /* ссылка на ячейку,
                                             содержащую "Фамилия" */
document.all.mytab.rows[1].cells[0]       /* ссылка на ячейку,
                                             содержащую "Иванов" */
document.all.mytab.rows[1].cells[1]       /* ссылка на ячейку,
                                             содержащую "Иван"*/
document.all.mytab.rows[2].cells[0]       /* ссылка на ячейку,
                                             содержащую "Петров" */
document.all.mytab.rows["r2"].cells[1]    /* ссылка на ячейку,
                                             содержащую "Петр" */
```

Выше мы рассмотрели создание ссылок на элементы таблицы. Эти ссылки применяются для чтения и изменения содержимого таблицы. В IE4+ для этой цели можно использовать свойства innerText и innerHTML С помощью свойства innerText читается и изменяется обычное текстовое содержимое ячеек. В случае когда в ячейке находится HTML-код, для его изменения необходимо аналогичным образом воспользоваться свойством innerHTML. Впрочем, если содержимое ячейки является обычным текстом, то вполне можно использовать и innerHTML. Таким образом, свойство innerHTML универсальнее, чем innerText. Если таблица выводится в окно браузера (то есть видима), изменение содержимого ее ячеек отобразится автоматически.

```
document.all.mytab.rows[1].innerText           // ИвановИванДиректор
document.all.mytab.cells[3].innerText          // Иванов
document.all.mytab.rows[1].cells[2].innerText  // Директор

/* Изменяем текстовое содержимое ячейки: */
document.all.mytab.rows[1].cells[2].innerText = "Сторож"
/* Заменяем текстовое содержимое ячейки на изображение: */
document.all.mytab.rows[1].cells[2].innerHTML = "<IMG SRC='pict.jpg'>"
```

Следует иметь в виду, что изменять содержимое таблицы необходимо по ячейкам, а не по строкам. Иначе говоря, попытка использовать выражение вида:

```
document.all.mytab.rows[индекс].innerText = новое_значение
```

приведет к удалению строки и, возможно, к ошибкам выполнения сценария.

Коллекции строк и ячеек, как и любой массив, обладает свойством length, значением которого является количество элементов в коллекции.

Примеры

```
document.all.mytab.rows.length            // количество всех строк
document.all.mytab.cells.length           // количество всех ячеек
document.all.mytab.rows[2].cells.length   /* количество ячеек в
                                             строке с индексом 2 */
```

4.8.2. Добавление и удаление строк таблицы

Добавление новой строки в таблицу производится с помощью метода insertRowQ. Этот метод возвращает ссылку на вновь созданную строку, которая затем используется для вставки ячеек. Ячейки вставляются в строку с помощью метода insertCell(индекс_ячейки). Данный метод возвращает ссылку на созданную ячейку, которая затем используется для задания содержимого ячейки. Ячейки вставляются в строку по порядку без пропусков начиная с нулевой. Так, нельзя создать ячейку

с индексом 3, если ячейки с индексами 0, 1 и 2 еще не созданы. С другой стороны, вы можете последовательно создать любое количество ячеек в строке.

Таким образом, новая строка создается в три этапа: сначала к таблице добавляется новая строка, затем в нее вставляются ячейки, после чего они заполняются значениями. Ниже приведен пример добавления строки к существующей таблице и заполнения ее ячеек значениями:

```
newrow = document.all.mytab.insertRow() // добавляем новую строку
newcell=newrow.insertCell(0)           // вставляем ячейку с индексом 0
newcell.innerText="Привет"             // заполняем ячейку значением
newcell=newrow.insertCell(1)
newcell.innerHTML="<IMG SRC='pict.jpg'>"
newcell=newrow.insertCell(2)
newcell.innerHTML="<B>Охранник</B>'>"
```

Для удаления строки таблицы служит метод deleteRow(индекс_строки). Параметр указывает номер удаляемой строки. Например:

```
document.all.mytab.deleteRow(2)
```

4.8.3. Генерация таблиц с помощью сценария

Один из неприятных моментов в создании таблиц посредством тегов HTML состоит в большом количестве этих тегов и необходимости тщательно следить за правильной их расстановкой. При этом коррекция содержимого ячеек больших таблиц оказывается весьма хлопотным делом. Облегчить работу в ряде случаев помогает хранение содержимого ячеек в массивах и генерация таблицы с помощью сценария.

В следующем примере предполагается, что содержимое строк таблицы хранится в отдельных массивах. С помощью операторов цикла формируется строка, содержащая теги таблицы, а затем эта строка записывается в HTML-документ.

```
<HTML>
<SCRIPT>
/* Массив заголовков столбцов */
ah = new Array ("Фамилия", "Имя", "Должность")
/* Массив данных */
ad = new Array()
   ad[0] = new Array( "Иванов", "Иван", "Директор")
   ad[1] = new Array( "Петров", "Петр", "Заместитель директора")
   ad[2] = new Array ("Сидорова", "Эллочка", "Секретарша")
   ad[3] = new Array ( "Федоров", "Федор", "Водитель")
strtab = "<TABLE>"
/* Формируем заголовки столбцов таблицы*/
for(i=0; i<ah.length; i++){
   strtab+= "<TH>" + ah[i] + "</TH>"
}
/* Формируем строки таблицы */
for(i=0; i<ad.length; i++) {
   strtab+= "<TR>"
   for(j=0; j<ad[i].length; j++){          // содержимое ячеек строки
      strtab+= "<TD>" + ad[i][j] + "</TD>"
   }
   strtab+="</TR>"
}
strtab+= "</TABLE>"
document.write(strtab)                    // запись в документ
</SCRIPT>
</HTML>
```

4.8.4. Простые базы данных

Достаточно эффективный способ формирования таблиц — использование специального элемента управления ActiveX, называемого Simple Tabular Data (STD — простые табличные данные). Этот элемент вставляется в HTML-документ с помощью тега <OBJECT> и позволяет легко управлять данными, хранящимися в обычном текстовом файле. Вы можете изменять, добавлять, удалять, сортировать, искать и фильтровать (выбирать) данные. Но главное достоинство применения STD заключается в простоте создания больших таблиц, например каталога ссылок.

Собственно данные хранятся на диске в текстовом файле. При этом их табличная структура поддерживается с помощью символов-разделителей. В качестве разделителя строк обычно используется невидимый символ перевода строки (клавиша Enter). Однако можно задать любой символ в качестве разделителя строк. Данные различных ячеек отделяются друг от друга произвольным символом, не используемым в каких-нибудь других служебных целях (например, запятая или вертикальная черта). Если запятая встречается в значениях данных, то ее не следует выбирать в качестве разделителя. Создавать такие файлы можно вручную с помощью текстового редактора (например, Блокнота Windows). Кроме того, в большинстве систем работы с базами данных возможно преобразование специфических файлов в текстовые.

В первой строке текстового файла записываются через символьный разделитель имена столбцов. Они не обязательно должны совпадать с теми заголовками столбцов, которые вы хотите видеть на экране. Главное их назначение в том, чтобы обеспечить доступ к данным. Это так называемые имена полей базы данных, или, другими словами, идентификаторы полей. Правила их образования такие же, как и для имен переменных. На рис. 4.26 первая строка содержит имена полей, а остальные — собственно данные. Разделителем полей здесь является вертикальная черта.

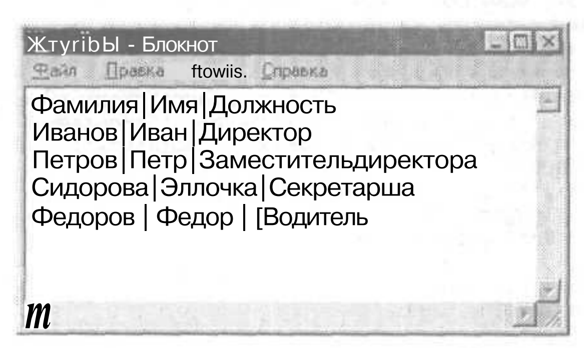


Рис. 4.26. Пример текстового файла с табличными данными

Элемент ActiveX STD, управляющий данными, встроен в браузер IE4+, а чтобы вставить его в HTML-документ, необходимо использовать следующие теги:

```
<OBJECT ID = "mydbcontrol"
   CLASSID = "CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">
   <PARAM NAME = "FieldDelim" VALUE="|">
```

```
    <PARAH NAME = "DataURL" VALUE = "mydb.txt">
    <PARAM NAME = "UseHeader" VALUE = true>
</OBJECT>
```

Контейнерный тег <OBЗECT> содержит множество тегов <PARAM>, с помощью которых задаются параметры. В примере мы указали лишь некоторые из них:

- FieldDelim — разделитель полей (ячеек); в примере значением этого параметра является вертикальная черта "|";
- DataURL— место расположения (URL-адрес) текстового файла сданными; в примере мы указали просто имя файла mydb.txt;
- UseHeader — определяет, содержит ли первая строка в текстовом файле имена полей.

Значение атрибута ID тега <OBJECT> выбирается произвольно, но оно обязательно должно быть указано. Если требуется установить символ, являющийся разделителем строк данных в текстовом файле, то следует добавить еще один параметр:

```
<PARAM NAME = "RowDelim" VALUE="&# ASCII-код;">
```

По умолчанию значением атрибута VALUE является "
",то есть ASCII-код символа перевода строки.

Для просмотра всех параметров элемента управления STD, установленных по умолчанию, достаточно загрузить в браузер следующий HTML-документ (рис. 4.27):

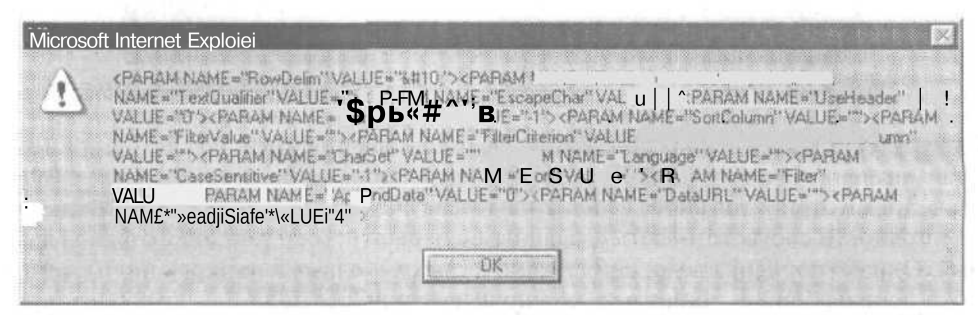


Рис. 4.27. Диалоговое окно, в котором показаны параметры элемента управления STD, установленные по умолчанию

```
<HTML>
<OBJECT ID = "mydbcontrol"
   CLASSIC = "CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">
</OBJECT>
<SCRIPT>
alert(mydbcontrol.tnnerHTML)
</SCRIPT>
</HTML>
```

Чтобы отобразить в окне браузера таблицу с данными из текстового файла mydb.txt, достаточно в том же документе записать следующий HTML-код (рис. 4.28):

```
<TABLE DATASRC = «mydbcontrol BORDER = 5>
<THEAD>
<TH>Фамилия сотрудника</TH><TH>Имя</TH><TH>Должность</TH>
</THEAD>
<TR>
```

```
<TD><SPAN DATAFLD = "Фамилия"></SPAN></TD>
<TD><SPAN DATAFLD = "Имя"></SPAN></TD>
<TD><SPAN DATAFLD = "Должность"></SPAN></TD>
</TR>
</TABLE>
```

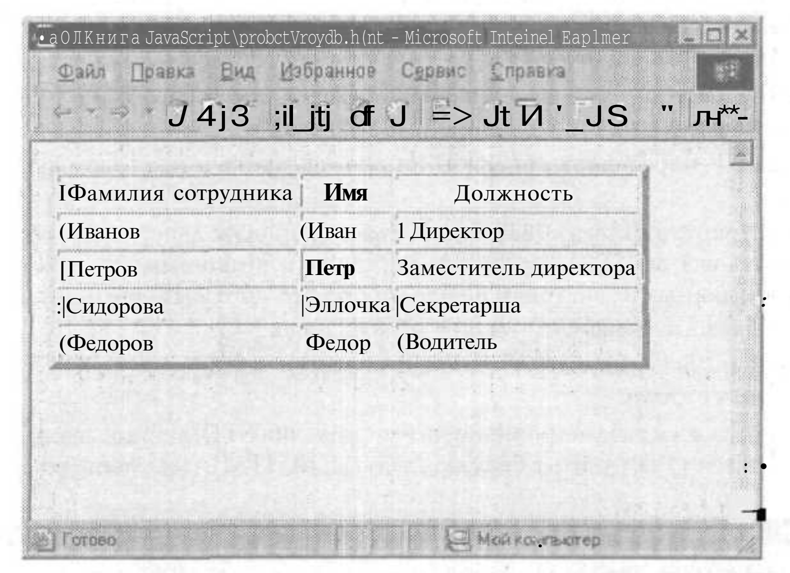


Рис. 4.28. Пример таблицы, созданной с помощью элемента управления STD

Здесь в теге <TABLE> атрибут DATASRC в качестве значения имеет ссылку на элемент STD (значение ID в теге <OBJECT>). Внутри тегов данных <TD>, в контейнерах <SPAN>, значениями атрибутов DATAFLD являются имена полей. Написание этих имен должно в точности соответствовать указанным в первой строке текстового файла данных. Однако порядок следования и количество выводимых на экран полей может быть произвольным. Например, можно вывести не все поля или, наоборот, продублировать некоторые из них. Кроме того, можно изменить порядок отображения полей. Этот порядок, а также состав отображаемых полей (столбцов) определяется в теге <TABLE>.

ВНИМАНИЕ

При использовании элемента управления STD описание таблицы с помощью тега <TABLE> имеет небольшой объем. В теге <TABLE>, по существу, описывается лишь структура таблицы, но не описывается ее содержимое. Описание данных содержится в текстовом файле.

Таким образом, при использовании STD мы организуем взаимодействие трех компонентов: элемент управления STD устанавливает связь с источником данных (текстовым файлом), элемент <TABLE> связывается с STD и источником данных, текстовый файл содержит собственно данные.

В текстовом файле с данными может содержаться не только текстовая информация, но и HTML-коды. Чтобы эти коды отображались в таблице не просто как текст, а интерпретировались, необходимо в теги <TD> добавить атрибут **DATAFORMATAS - "html".**

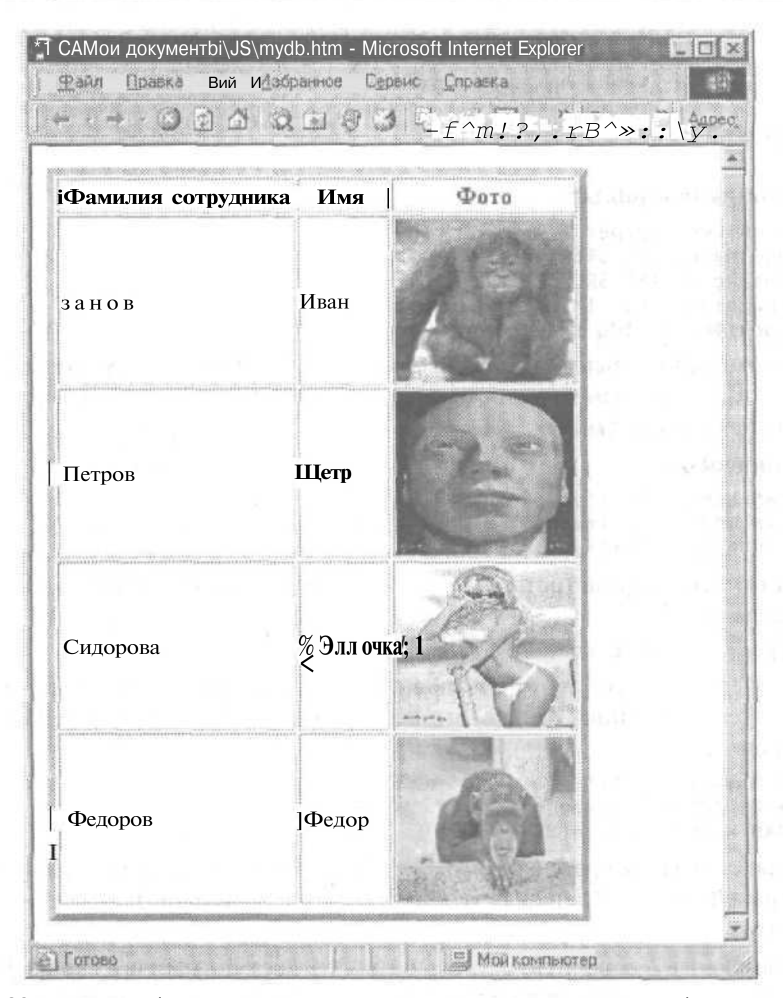


Рис. 4.29. Пример таблицы, источник данных которой содержит теги изображения <IMG>

При этом если ячейка таблицы не содержит тегов, то будет отображаться простой текст, а в противном случае — результат выполнения HTML-кода. Это позволяет, например, вставлятьвячейкитаблицыизображения, гиперссылки, кнопкиидругие элементы (рис. 4.29). Ниже приведен пример:

```
<HTML>
<OBJECT ID = "mydbcontrol"
CLASSID = "CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">
<PARAM NAME = "FieldDelim" VALUE="|">
<PARAM NAME = "DataURL" VALUE = "mydb.txt">
<PARAM NAME = "UseHeader" VALUE = true>
</OBJECT>
<TABLE DATASRC = tfmydbcontrol BORDER = 5>
<THEAD>
<TH>Фамилия сотрудника</TH><TH>Имя</TH><TH>Фото</TH>
</THEAD>
<TR>
```

```
<TD><SPAN DATAFLD = "Фамилия"></5PAM>< /TO>
<TD><SPAN DATAFLD = " Имя " >< / S PAN></TD>
<TD><SPAN DATAFLD = "Портрет" DATAFORMATAS = " html" > </ SPAN></TD>
</TR>
</TABLE>
</HTML>
```

Файл с данными mydb.txt:

```
Фамилия|Имя|Портрет
Иванов|Иванj<IMG SRC='pictl.jpg'>
Петров | Петр | <IMG SRC=' pi ct2 .jpg'>
Сидорова|Эллочка|<IMG SRC='pict3.jpg'>
Федоров|Федор|<IMG SRC='pict4.jpg'>
```

Как уже отмечалось выше, элемент управления STD имеет и другие параметры. В частности, есть параметры, с помощью которых можно установить фильтр (выборку) и сортировку данных.

Для установки фильтра предусмотрены следующие три параметра:

```
<PARAM NAME = "FiUerColumn" VALUE = "имя_поля"
<PARAM NAME = "Fi1terCriterion" VALUE = "оператор_сравнения"
<PARAM NAME = "FilterValue" VALUE = образец
```

С помощью этих параметров задается логическое условие (критерий) фильтра следующего вида:

```
имя_поля оператор_сравнения образец
```

Например, если требуется отфильтровать из таблицы все строки, в которых поле Фамилия имеет значение "Иванов", то необходимо использовать следующие теги параметров:

```
<PARAM NAME = "Filter-Column" VALUE = "Фамилия"
<PARAM NAME = "FilterCriterion" VALUE = "=="
<PARAM NAME = "FilterValue" VALUE = "Иванов"
```

В качестве оператора сравнения можно использовать = (неточное равенство), == (точное равенство), != (неравенство), > (больше), < (меньше), >= (не больше), <= (не меньше).

Чтобы выключить (сбросить) фильтр и сделать доступными все строки таблицы, достаточно использовать условие, которому заведомо удовлетворяют все строки таблицы. Например, следующие параметры сбрасывают текущий фильтр, поскольку пустая строка содержится в любой строке:

```
<PARAM NAME = "FilterCriterion" VALUE = "="
<PARAM NAME = "FilterValue" VALUE = '"
```

По умолчанию фильтрация данных чувствительна к регистру, в котором набран образец. Управлять зависимостью от регистра можно с помощью параметра CaseSensitive:

```
<PARAM NAME = "CaseSensi tive" VALUE = значение>
```

Атрибут VALUE может принимать значения O/fatse (не зависит) или I/true (зависит).

Для сортировки (упорядочения) данных используется следующий параметр:

```
<PARAM NAME = "SortColumn" VALUE="HMfl_nona">
```

Сортировка строк производится по значениям ASCII-кода значений заданного поля базы данных. Например, если мы хотим, чтобы фамилии располагались по алфавиту, необходимо записать:

<PARAM NAME = "SortColumn" VALUE = "Фамилия">

По умолчанию сортировка производится по увеличению значения ASCII-кода символов. Однако порядок можно изменить на противоположный, если использовать следующий параметр:

<PARAM NAME="SortAscending" VALUE=0>

Атрибут VALUE=l устанавливает режим сортировки, принятый по умолчанию.

Элемент управления STD предоставляет объект recordset, который поддерживает методы перемещения по строкам таблицы (по записям базы данных). Синтаксис следующий:

1c1_объекта. recordset ,метод

Например, mydb.recordset.moveNextQ перемещает указатель текущей записи на следующую запись.

Методы перемещения по записям базы данных:

- movePreviosQ — к предыдущей записи;
- moveNextQ — к следующей записи;
- moveFirstQ— к первой записи;
- moveLastQ — к последней записи.

Перед использованием методов moveNextQ и movePreviosQ следует проверять значения свойств eof (конец файла данных) и bof (начало файла данных). Так, если текущей является последняя строка таблицы, то нельзя использовать метод moveNextQ. Аналогично, если текущей является первая строка, то нельзя использовать movePreviosQ. Разумеется, методы перемещения используются в сценариях обработки событий, таких как, например, щелчок на кнопке. Это необходимо, если на экран выводятся не все записи (строки) таблицы, а только одна запись



Рис. 4.30. Пример формы с полями ввода и просмотра данных и кнопками для перемещения по записям

(например, с помощью полей ввода). Перемещение по строкам таблицы используется при поиске данных в соответствии с некоторым критерием и при их обработке.

Если использовать поля ввода (например, при создании формы для просмотра и коррекции данных), то привязку данных к ним можно осуществить с помощью следующего фрагмента HTML-кода:

```
<TR><TD>
<INPUT TYPE = "text" DATASRC = #mydbcontrol DATAFLD = "Фамилия">
</TD></TR>
```

Обратите внимание, что атрибуты привязки данных DATASRC и DATAFLD совместно используются в одном и том же теге. В листинге 4.20 приводится пример HTML-кода, который выводит на страницу одну запись базы данных с кнопками навигации (рис. 4.30). Здесь использовался тот же источник данных, что и в предыдущем примере.

Листинг 4.20. Код, выводящий одну запись базы данных с кнопками навигации

```
<HTML>
<HEADER><TITLE>npHMep формы STD</TITLE></HEADER>
<OBJECT ID = "mydbcontrol"
 CLASSID = "CISID:333C7BC4-460F-11D0-BC04-0080C7055A83">
<PARAM NAME = "FieldDelim" VALUE = "|">
<PARAM NAME = "DataURL" VALUE = "mydb.txt">
<PARAM NAME = "UseHeader" VALUE = true>
</OBJECT>

<! Поля с данными>
<TABLE WIDTH = 75%>
<TR><TH><t>aMHnHfl</TH>
<TD>
<INPUT TYPE = "text" DATASRC = #mydbcontrol DATAFLD = "Фамилия" >
</TD></TR>
<TИ><TH>Имя</TH>
<TD><INPUT TYPE = "text" DATASRC = #mydbcontrol DATAFLD="HMfl">
</TD></TR>
<TK><TH>Фото</TH>
<TD><SPAN DATASRC = #mydbcontrol DATAFLD = "Портрет" DATAFORMATAS =
  "html"></SPAN>
</TD></TR>
</TABLE>
<P>
<! Кнопки перемещения по записям
<INPUT NAME = "cmdFirst" TYPE = "BUTTON" VALUE = "«"
  onclick = "FirstO">
<INPUT NAME = "cmdPrevious" TYPE="BUTTON" VALUE="<"
  onclick = "PreviousO">
<INPUT NAME="cmdNext" TYPE="BUTTON" VALUE=">"
  onclick = "Next()">
<INPUT NAME="cmdLast" TYPE="BUTTON" VALUE=">>"
  onclick = "LastO">

<SCRIPT>
function FirstO {                    // к первой записи
mydbcontrol.recordset .moveFirst()
}
```

```
function PreviousO {                          // к предыдущей записи
{
if (! mydbcontrol.recordset.bof)
mydbcontrol.recordset.movePreviousO
}
function Next() {                             // к следующей записи
if (! mydbcontrol.recordset.eof)
mydbcontrol.recordset.moveNextO
}
function Last() {                             // к последней записи
mydbcontrol.recordset.moveLast()
}
</SCRIPT>
</HTML>
```

Обратите внимание, что третье поле с именем Портрет содержит изображения, и поэтому мы показываем его не с помощью элемента <INPUT>, а с помощью контейнера <SPAN> с атрибутом DATAFORMATAS = "html".

ВНИМАНИЕ -

Изменения данныхвтаблицесохраняютсятольковоперативной памяти компьютера и никак не влияют на содержимое текстового файла-источника данных. Чтобы обновить содержимое таблицы, необходимо отредактировать текстовый файл и перекачать его на сервер, как это обычно делается при модификации страниц сайта.

Тем не менее рассмотренный здесь элемент управления STD оказывается очень полезным для организации обновления содержания веб-сайта. Вместо того чтобы корректировать HTML-код всей страницы, выискивать в ней нужное место, вы можете редактировать только текстовые файлы с информацией, не занимаясь при этом вопросами композиции. Например, таким образом можно организовать разделы новостей сайта и текущей информации. В подразделе 4.8.7 будет рассмотрена система поиска по сайту, основанная на использовании STD.

4.8.5. Сортировка данных таблицы

Рассмотрим сортировку строк таблицы по значениям того или иного столбца. В нашем примере для сортировки необходимо просто щелкнуть на заголовке нужного столбца таблицы. Для этого кроме элемента управления STD и таблицы нам понадобится сценарий, содержащий функцию, обрабатывающую щелчок путем модификации тега <OBJECT> (листинг 4.21). Здесь использован следующий хитрый прием. Сначала база данных показывается без всякой сортировки. Мы сохраняем блок тегов от <OBJECT> до </OB3ECT>, за исключением последнего, в переменной obj. При щелчке на заголовке таблицы вызывается функция сортировки sort(field), которой передается имя поля. Эта функция дописывает к значению переменной obj строку, содержащую теги параметров, отвечающие за сортировку. Далее функция sort() заменяет в HTML-коде теги <OBЖT> на те, которые содержатся в переменной obj. Это делается с помощью свойства outerHTML При этом элемент управления STD заново инициализируется с новыми параметрами. В результате таблица перерисовывается в соответствии с новым порядком следования строк.

Листинг 4.21. Код сценария сортировки данных таблицы

```
<HTML>
<OBJECT ID = "mydbcontrol"
CLASSID = "CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">
<PARAM NAME '  FieldDelim" VALUE="|">
<PARAM NAME '  DataURL" VALUE = "mydb.txt">
<PARAM NAME = 'UseHeader" VALUE = true>
<PARAM NAME = ' "SortColumn"  VALUE"Фамилия">
<PARAM NAME - 'SortAscending" VALUE = 1>
</OBJECT>

<TABLE DATASRC = tfmydbcontrol BORDER = 5>
<THEAD>
<TH onclick = "sort('Фамилия')">Фамилия сотрудника</TH>
<TH onclick = "sort('Имя') ">Имж/TH>
<TH>Фото</TH>
</THEAD>
<TR>
<TD><SPAN DATAFLD = "Фамил ия " ></ SPAN></TD>
<TD><SPAN DATAFLD = " Имя " >< / SPAN></TD>
<TD><SPAN DATAFLD = "Портрет" DATAFORMATAS = "html"> </SPAN></TD>
</TR>
</TABLE>

<SCRIPT>
// Сохраняем теги объекта SID в переменной
var x = document.all.mydbcontrol.innerHTML  // теги, вложенные в
<object>
var obj = '<OBJECT ID = "mydbcontrol" CLASSID =
  "CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">' + x

function sort(field) {    // сортировка по значениям столбца field
var y = document.all.mydbcontrol
y.outerHTML = obj + '<PARAMNAME = "SortColumn" VALUE"' + field +
  '"></OBJECT>'
}
</SCRIPT>
</HTML>
```

В этом примере сортировать данные можно только по первым двум текстовым полям. Вы можете в качестве упражнения усовершенствовать этот код. Например, сделайте так, чтобы при двойном щелчке на заголовке столбца таблицы сортировка происходила в противоположном порядке. Для этого необходимо не устанавливать параметр <PARAM NAME="SortAscending" VALUE=0>.

4.8.6. Фильтрация данных таблицы

Для организации фильтрации данных из таблицы по заданному критерию используется подход, аналогичный рассмотренному в предыдущем подразделе. В приведенном ниже примере (листинг 4.22) в окно браузера выводится таблица с данными и элементы, с помощью которых можно задать поле (столбец) и значение (образец) для формирования простого условия фильтра вида: имя_поля = значение.

Имя поля выбирается из раскрывающегося списка, а значение вводится с клавиатуры. Для установки фильтра следует щелкнуть на кнопке Применить (рис. 4.31).

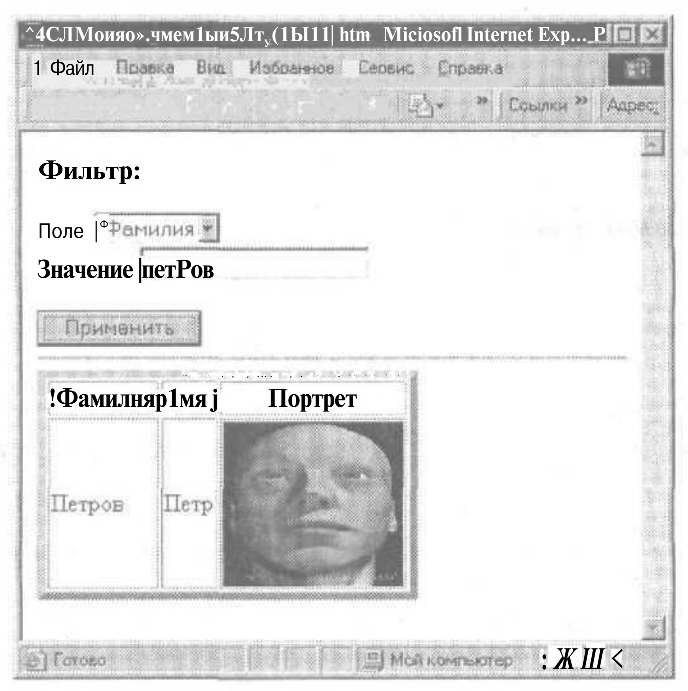


Рис. 4.31. Пример таблицы с элементами для установки фильтра

Если поле ввода значения пусто, то при щелчке на этой кнопке фильтр выключится, а в таблице будут отображены все имеющиеся записи. Заметим, что результат фильтрации в данном примере не зависит от регистра, в котором пользователь ввел значение.

Также, как и при сортировке, мы сохраняем в переменной часть информации о параметрах элемента STD, а затем модифицируем ее, чтобы учесть условие фильтра. Для установки фильтра используется свойство outerHTML

Листинг 4.22. Код для фильтрации данных таблицы

```
<HTML>
<H3>Фильтр:</H3>
Поле
<! Раскрывающийся список имен полей >
<SELECT NAME = "FLD">
<OPTION value = "Фамилия">Фамилия
<OPTION value = "Имя">Имя
</SELECT>
<BR>
Значение
<! Поле ввода значения для фильтра и кнопка>
<INPUT NAME = "IMP" VALUE = "" TYPE = "text">
<P>
<BUTTON onclick = "fiПегО">Применить</B1ЛТ0M>
<HR>

<! Элемент управления STD>
```

продолжение

Листинг 4.22 *(продолжение)*

```
<OBJECT ID = "mydbcontrol"
CLASSID = "CLSID:333C7BC4-460F-11D0-BC04-0080C7Q55A83">
<PARAM NAME ="FieldDelim" VALUE="|">
<PARAM NAME = "DataURL" VALUE = "mydb.txt">
<PARAM NAME = "UseHeader" VALUE = true>
</OBJECT>
<! Таблица для вывода данных>
<Table DATASRC = tfmydbcontrol border = 5>
<THEAD>
<TH>Фамилия</TH>
<TH>Имж/TH>
<TH>Портрет</TH>
</THEAD>
<TR>
<TD><SPAN DATAFLD = "Фамилия"></5PAM></TO>
<TD><SPAN DATAFLD = "Имя" ></5PAN></TD>
<TD><SPAN DATAFLD = "Портрет" DATAFORMATAS="html "></SPAN></TD>
</TR>
</TABLE>
<SCRIPT>
/* сохраняем основные параметры элемента SID в переменной obj */
var obj = "<OBJECT ID = 'mydbcontrol' "
obj+= "CLASS ID='CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83'>"
obj+="<PARAM NAME='FieldDelim' Value = ')'>"
obj+= "<PARAM NAME='DataURL' Value = 'mydb.txt'>"
obj+= "<PARAM NAME=' UseHeader' value = true>"

function filter() {            // установка фильтра
var cpar = ""                  // переменная для хранения параметров
фильтра
if (INP.value){                // если введено значение
   cpar="<param name='FilterColumn' value = '"+FLD.value+"'>"
   cpar+= "<param name='FilterValue' value = '"+INP.value +"'>"
   cpar+= "<param name='FilterCriterion' value = '=' >"
   cpar+="<PARAM NAME='CaseSensitive' VALUE = false>"
}
document.all.mydbcontrol.outerHTML = obj + cpar + "</OBJECT>"
}
</SCRIPT>
</HTML>
```

Для тренировки усовершенствуйте рассмотренный выше код таким образом, чтобы можно было выбирать оператор сравнения из раскрывающегося списка, а также устанавливать режим зависимости или независимости от регистра.

4.8.7. Поиск по сайту

Если ваш сайт содержит много страниц, то имеет смысл организовать поисковую систему. Здесь мы рассмотрим один простой вариант такой системы, использованный на моем сайте.

В основе поисковой системы положена база данных, содержащая соответствие между поисковыми образами (ключевыми словами) и ссылками на веб-страницы. Эта поисковая база данных создана в обычном текстовом файле, который в нашем примере называется search.txt. В ней всего два поля (столбца) с именами p!

и p2. Первый столбец содержит ключевые слова, а второй — ссылки на HTML-документы, в которых находятся соответствующие ключевые слова. В качестве разделителя полей в текстовом файле использовалась вертикальная черта. Таким образом, структура поисковой базы данных имеет следующий вид:

```
p1|p2
ключевые_слова|ссылки
```

Ниже приводится фрагмент поисковой базы данных для моего сайта:

```
p1|p2
БД текстовая база данных STD Simple Tabular Data|<a href='db0.htm'>База данных (STD)</a>
Динамическая загрузка элементов|<a href='change.htm'>Динамическая загрузка элементов</a>
E-mail Email Электронная почта|<a href='mail.htm'>Электронная почта</a>
Эллиптичекая траектория орбита движение эллипс|<a href='ellipse.htm'>Движение по эллипсу</a>
Звездные войны Космос|<a href='ellipse.htm'>Звездные войны</a>
Подсветка мигание бордюр|<a href='bordur.htm'>Подсветка и мигание</a>
Поиск по сайту карта сайта map search|<a href='search.htm'>Поиск по сайту</a>
Поиск в тексте поиск в текстовой области|<a href='findtext.htm'>Поиск в текстовой области</a>
АФП AFP|<a href='afp.htm'>АФП</a>
выполнение выражения eval()|<a href='eval.htm'выполнение выражения в текстовой строке</a>
книги книга Сам себе web-дизайнер web-мастер|<a href='mybook.htm'>Книги</a>
```

Здесь ключевые слова выбраны из заголовков и содержательной части текстов, расположенных на различных страницах сайта, посвященного веб-дизайну. Например, такие слова, как «БД», «текстовая», «база» и «данных», содержатся в файле **db0.htm.** Хотя эти же слова можно встретить и на других страницах моего сайта, я сделал так, чтобы поиск этих слов приводил именно к документу db0.htm.

Прежде всего, необходимо тщательно продумать систему ключевых слов, чтобы поиск был эффективным. Затем следует разработать алгоритм работы поисковой системы. В частности, требуется решить, будет ли поиск зависеть от регистра, в котором пользователь ввел поисковый образ, выводить ли в качестве результата первый найденный документ или выводить только список ссылок на все найденные документы и т. п.

В рассматриваемом примере поиск не зависит от регистра введенного поискового образа. Система просматривает значения первого поля базы данных строка за строкой, проверяя каждый раз, содержится ли введенный пользователем поисковый образ в значении первого поля. Если да, то поиск завершается; в противном случае он продолжается. Результат поиска выводится в виде ссылки на документ. Щелчок на данной ссылке выводит этот документ в окно браузера. Если поиск неудачен, то появляется соответствующее сообщение.

Интерфейс поисковой системы прост: поле ввода поискового образа и кнопка для запуска процедуры поиска. Результаты поиска отображаются ниже, под полем ввода (рис. 4.32).

Для работы с базой данных использованы два идентичных элемента управления **STD,** связанные с одним и тем же текстовым файлом search.txt. К каждому из этих элементов STD привязана своя таблица. Одна из них, всегда невидимая, необходима для служебных целей, а именно для операции поиска по всей базе данных. Она содержит только поле ключевых слов p1. Вторая таблица предназначена для отображения результатов поиска и содержит только поле ссылок p2. При загрузке документа эта таблица не видна и отображается только при успешном поиске. Такой прием обусловлен тем, что стандартные для элемента **STD** средства фильтрации (операторы сравнения) слишком примитивны для наших целей (см. подраздел 4.8.4). Нам было необходимо организовать проверку

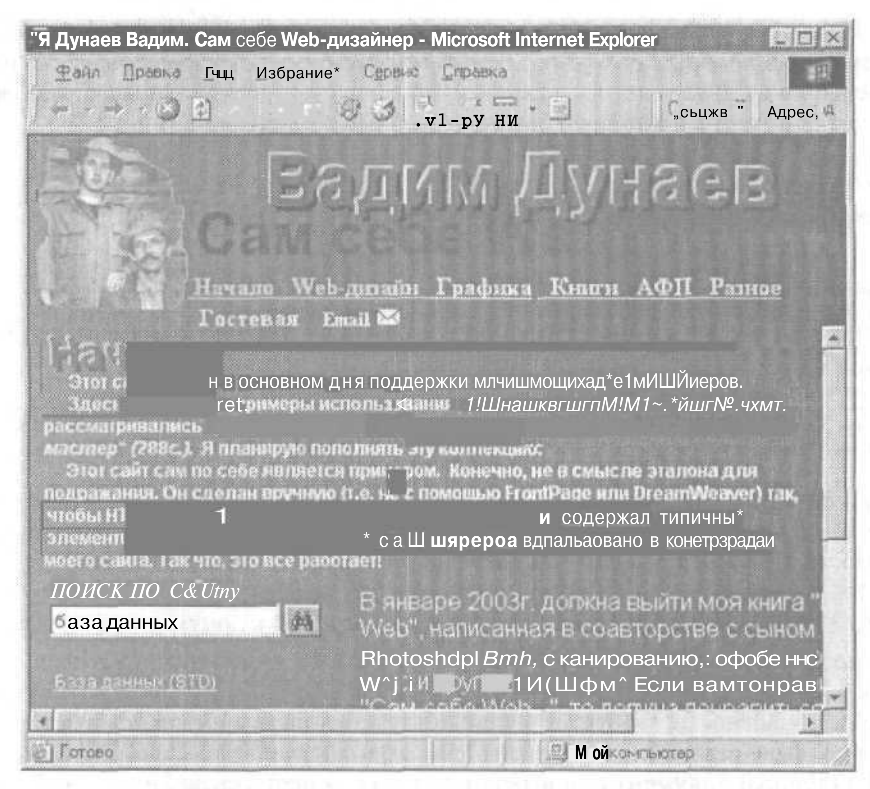


Рис. 4.32. Так выглядит интерфейс поисковой системы на главной странице сайта автора

вхождения поискового образа в последовательность ключевых слов. Очевидно, что это недостижимо с помощью обычных операторов сравнения. Кроме того, мне не хотелось сбрасывать фильтр перед выполнением каждой операции поиска. В листинге 4.23 приводится пример кода поисковой системы.

Листинг 4.23. Пример кода поисковой системы

```
<HTML>
<! Элемент управления для таблицы ссылок >
<OBJECT ID = 'fnd1'
   CLASSID = 'CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83'>
   <PARAM NAME = 'FieldDelim' VALUE='|'>
   <PARAM NAME = 'DataURL' VALUE='search.txt'>
   <PARAM NAME = 'UseHeader' VALUE=true>
   <PARAM NAME = 'CaseSensitive' VALUE=false>
</OBJECT>

<! Элемент управления для таблицы ключевых слов >
<OBJECT ID = 'fnd2'
   CLASSID = 'CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83'>
   <PARAM NAME = 'FieldDelim' VALUE='|'>
   <PARAM NAME = 'DataURL' VALUE='search.txt'>
   <PARAM NAME = 'UseHeader' VALUE=true>
   <PARAM NAME = 'CaseSensitive' VALUE=false>
   <PARAM NAME = 'CharSet' VALUE='windows-1251'>
```

```
</OBJECT>

<! Интерфейс поисковой системы >
<! Поле ввода поискового образа >
<INPUT TYPE = "text" NAME = "WORD" VALUE = "" SIZE = 20>
<! Кнопка запуска процедуры поиска >
<BUTTON onclick = "findkeyword()">поиск</BUTTON><BR>
<! Здесь будет выводиться результат поиска>
<B ID = 'rezult'></B>

<! Таблица ключевых слов >
<TABLE STYLE = 'visibility:hidden'>
<TR>
<TD><INPUT NAME = 'fl' DATASRC = #fnd2 DATAFLD = 'pl'x/TD>
</TR>
</TABLE>

<! Таблица ссылок >
<TABLE ID = 'mytab' DATASRC = #fndl WIDTH = 350 ALIGN = LEFT
STYLE = 'visibility:hidden'>
<TR>
<TD><SPAN DATAFLD = 'p2' DATAFORMATAS = 'html'></SPAN></TD>
</TR>
</TABLE>

<SCRIPT>
var obj = "<OBJECT ID = fndl "
obj+="CLASSID='CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83'>"
obj+="<PARAM NAME='FieldDelim' VALUE='r>"
obj+="<PARAM NAME='DataURL' VALUE='search.txt'>"
obj+="<PARAM NAME='UseHeader' VALUE=true>"
obj+="<PARAM NAME='CaseSensitive' VALUE=false>"

function findkeyword() {              // поиск по введенному образцу
if (!WORD.value) return               // если поисковый образ не введен
var xfltr = "", y, cpar
fnd2.recordset.moveFirst()            /* переход к 1-й записи таблицы
                                         с ключевыми словами */
while (!fnd2.recordset.eof){          // пока не достигли конца таблицы
   y = fl.value.toLowerCase()
   if (y.indexOf(WORD.value.toLowerCase())>=0) { // проверка на вхождение
      xfltr = fl.value                       /* значение для
                                                формирования
                                                условия фильтра */
            break
   I
   fnd2.recordset.moveNext()          /* переход к следующей записи
                                         таблицы с ключевыми словами */
}
if (!xfltr)
   document.all.rezult.innerText = "Ничего не найдено"
else
   document.all.rezult.innerText = "

/* Формируем условие фильтра: */
cpar = obj + "<param name = 'FilterColumn' value = 'pl'>"
cpar+= "<paramname = 'FilterCriterion' value = '='>"
cpar+= "<param name = 'FilterValue' value = '" + xfltr + "'></OBJECT>"
```

продолжение ↗

Листинг 4.23 *(продолжение)*

```
document.all.fndl.outerHTML = cpar    /* устанавливаем фильтр
                                         в таблице ссылок */
mytab.style.visibility = "visible"    // делаем таблицу ссылок видимой

</SCRIPT>
</HTML>
```

Функция поиска findkeyword() сканирует записи базы данных, проверяя для каждой из них, содержит ли поле ключевых слов p1 введенный пользователем поисковый образ. Если результат проверки положителен, то переменной xfltr присваивается значение поля ссылок p2. Это значение затем используется для формирования условия фильтра, а сканирование базы данных прекращается. Установка фильтра производится с помощью присвоения нового значения свойству outerHTML элемента управления STD для таблицы ссылок. После установки фильтра мы делаем таблицу ссылок видимой. Эта таблица будет содержать лишь одну ссылку.

Итак, мы рассмотрели простой вариант поисковой системы. Его можно усовершенствовать по нескольким направлениям. Мы рассмотрим далее лишь одно из них. А именно, давайте сделаем так, чтобы система поиска сканировала все записи базы данных и выводила в качестве результата все ссылки, соответствующие поисковому образу, а не только одну первую найденную. Для осуществления этого замысла нам потребуется один элемент управления STD и одна невидимая таблица с двумя полями, p1 и p2. Фильтрация данных нам не понадобится. Вместо нее мы просто будем формировать строку, содержащую найденные ссылки, которую затем присвоим свойству innerHTML объекта result. При этом код даже упрощается по сравнению с исходным (листинг 4.24).

Листинг 4.24. Код усовершенствованной поисковой системы

```
<HTML>
<! Элемент управления >
<OBJECT ID = 'fndl'
   CLASSID = 'CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83'>
   <PARAM NAME = 'FieldDelim' VALUE='|'>
   <PARAM NAME = 'DataURL' VALUE='search.txt'>
   <PARAM NAME = 'UseHeader' VALUE=true>
   <PARAM NAME = 'CaseSensitive' VALUE=false>
</OBJECT>
<! Интерфейс поисковой системы >
<! Поле ввода поискового образа >
<INPUT TYPE = "text" NAME = "WORD" VALUE = "" SIZE = 20>
<! Кнопка запуска процедуры поиска >
<BUTTON onclick = "findkeyword()">Поиск</BUTTON><BR>
<! Здесь будет выводиться результат поиска>
<B ID = 'rezult'></B>

<! Таблица поисковой базы данных >
<TABLE STYLE ='visibility:hidden'>
<TR>
<TD><INPUT NAME = 'f1' DATASRC = #fndl DATAFLD = 'pi' ></TD>
<TD><INPUT NAME = 'f2' DATASRC = #fndl DATAFLD = 'p2'
 DATAFORMATAS = "html"></TD>
</TR>
</TABLE>
```

```
<SCRIPT>
function findkeyword() {            // поиск по введенному образцу,
if (IWORD.value) return             // если поисковый образ не введен
var xrezult = "", y
fnd1.recordset.moveFirst()          // переход к 1-й записи таблицы
while (!fndl.recordset.eof){        // пока не достигли конца таблицы
   y = fl .value.toLowerCaseO
   if (y.indexOf(WORD.value.toLowerCase( ))>=0){ // проверка на вхождение
           xrezult+= f2.value +"<BR>"            // формируем строку ссылок
x
1  fnd1.recordset.moveNext()
                                    // переход к следующей записи таблицы
}
 if (Ixrezult) document.all.rezult.innerHTML = "Ничего не найдено"
 else
      document.all. rezult. innerHTML = xrezult
I
</SCRIPT>
</HTML>
```

4.8.8. Вставка HTML-документа в таблицу

Документ из внешнего HTML-файла можно вставить в таблицу, находящуюся в текущем документе и управляемую элементом STD. Эта возможность основана на том, что HTML-файл является обычным текстовым файлом. Поскольку в первой строке этого файла указывается тег<HTMl_>, мы можем использовать его в качестве имени поля базы данных. Признаюсь, эта простая мысль не сразу пришла мне в голову. Видимо, сказался стереотип образования имен переменных.

Таким образом, мы рассматриваем вставляемый HTML-файл как базу данных с единственным полем (столбцом). Основное требование к содержимому HTML-файла: любой контейнерный тег должен полностью размещаться в одной строке этого файла.

Рассмотрим сначала пример вставки в таблицу одного HTML-документа:

```
<HTML>
<! Элемент управления >
<OBJECT ID = 'mydbcontrol'
   CLASSID = 'CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83'>
   <PARAM NAME = 'FieldDelim' VALUE='|'>
   <PARAM NAME = 'DataURL' VALUE='mydocum.htm'>
   <PARAM NAME = 'UseHeader' VALUE=true>
</OBJECT>
<TABLE DATASRC = #mydbcontrol WIDTH = 350>
<TR>
<TD><SPAN DATAFLD = '<HTML>' DATAFORMATAS = 'html'></SPAN></TD>
</TR>
</TABLE>
</HTML>
```

Здесь предполагается, что в документе тег <HTML> записан прописными буквами. Если же он записан строчными буквами, то это должно быть отражено и в значении атрибута DATAFLD = "<html>". Очевидно, вы можете задать атрибуты оформления таблицы (рамок, выравнивания, цвета и т. п.), а также стилевые параметры позиционирования.

Теперь рассмотрим случай вставки нескольких HTML-документов. В листинге 4.25 приведен сценарий, который это выполняет.

Листинг 4.25. Код сценария вставки нескольких HTML-документов

```
•<SCRIPT>
/* Вставка HTML-документов */
/* Массив имен (URL) вставляемых HTML-файлов */
var aurl = new ArrayO
aurl[0] = "htmll.htm"
aurl[1] = "html2.htm"
var tabwidth=800;             /* ширина таблицы, в которой будут показаны
                                 HTML-документы */
/* Массив частей строки, определяющей элемент управления STD */
var xobject = new ArrayO
xobject[0] = '<OBJECT ID="idobj'        //id
xobject[1] = '" CLASSID="CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">'
xob]'ect[l] += '<PARAMNAME = "FieldDelim" VALUE = "|">'
xobjectfl] += '<PARAMNAME = "UseHeader" VALUE = tme>'
xobject[1] += '<PARAMNAME = "CharSet" VALUE = "windows-1251">'
xobject [1] += '<PARAMNAME = "DataURL" VALUE ="'
xobject[2] = '"></OBJECT>'

/* Строка тегов таблицы */
var xtab = new ArrayO
xtab[0] = '<TABLE WIDTH=' + tabwidth + ' DATASRC = #'
xtab[1] = 'XTRXTDXSPANDATAFLD="<ritml>" DATAFORMATAS= " html" > '
xtab[l] += '</SPAN></TD></TRx/TABLE>'
var sobj = "";
for (i =0; i < aurl.length; i++) {
sobj += xobject[0] + i + xobjectll] + aurl[i] + xobject[2] + xtab[0] +
   'idobj ' + i + xtabfl]
}
document.write(sobj)
</SCRIPT>
```

Для пробы возьмем следующие два простых HTML-документа:

Файл htmll.htm:

```
<HTML>
<НЗ>Документ 1</НЗ>
<image src="folder.gif">3To - просто картинка
<a href="db0.htm">Описание баз данных в текстовых файлах</a>
<button onclick="alert('Привет!!!')">Щелкни</button>"Это - кнопка
</HTML>
```

Файл html2.htm:

```
<HTML>
<НЗ>Документ 2</НЗ>
<1>Это - содержание документа из второго файла.<1>
В нем только этот текст
</HTML>
```

Тогда HTML-документ, содержащий лишь приведенный выше сценарий, будет выглядеть в окне браузера, как показано на рис. 4.33. Диалоговое окно с приветствием появляется при щелчке на кнопке Щелкни.

Вы можете разместить в основном документе элементы управления (ссылки, кнопки), с помощью которых в одну и ту же таблицу будут загружаться различные

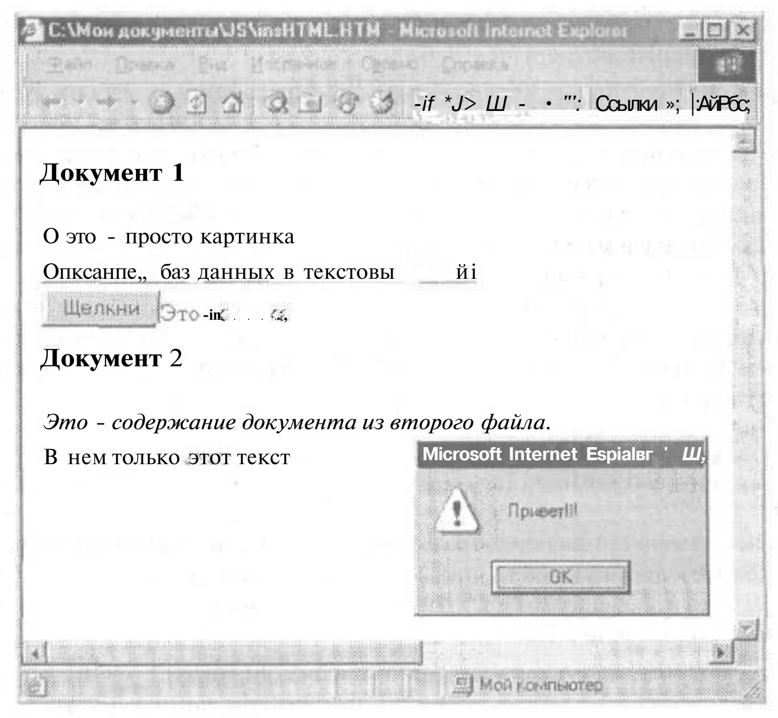


Рис. 4.33. Результат вставки HTML-документов втаблицы

HTML-документы. Для этого необходимо соответствующим образом изменять значение параметра DataURL.

Рассмотренный выше способ является еще одним средством (помимо плавающих фреймов <IFRAME>) вставки HTML-документов, осуществляемой на стороне клиента, то есть браузером пользователя, а не сервером.

ВНИМАНИЕ

В отличие от плавающихфреймов, этот способ не столь универсален. При его использовании следуетучитывать, в каком регистре записан тег <HTML> во вставляемом документе, атакже следить затем, чтобы контейнерные теги (а их подавляющее большинство в HTML) записывались целиком в одной строке (без переносов).

4.8.9. Обработка табличных данных

Нередко в базах данных хранят некоторые первичные данные, а отображают на экране (или используют как-то иначе) результаты их обработки. Обычно обработке подвергаются числовые данные. Например, требуется вычислить сумму всех значений столбца таблицы или найти максимальное или минимальное значение. Бывают и более сложные задачи. В любом случае для их решения удобно, прежде всего, считать обрабатываемые данные из таблицы в массивы, для которых в JavaScript предусмотрены многочисленные методы обработки. Дополнительно

к этим методам можно создать свои специальные функции обработки. Некоторые из них были рассмотрены в подразделе 1.7.2.

Рассмотрим задачу считывания значений столбца таблицы в массив. Если таблица с данными полностью создана с помощью тегов HTML (без использования элемента управления STD), то задача считывания проста. В этом случае требуется с помощью оператора цикла присвоить элементам массива значения ячеек таблицы из заданного столбца. Пусть, например, элемент <TABl_E> имеет ID = "mytab", и требуется считать в массив все значения столбца с индексом n. Далее предположим, что содержимым ячеек этого столбца являются числа, и мы собираемся производить числовые операции над элементами массива. Поскольку содержимое ячеек таблицы имеет строковый тип, постольку при считывании сразу же воспользуемся функцией parseFloatQ или parselntQ. Ниже приведен сценарий, решающий поставленную задачу:

```
myarray = newArrayO
for(i = 0; i< document.all.mytab.rows.length; i++){
myarray[i] = parseFloat(document.all.mytab.rows[i].cells[n])
}
```

Здесь мы использовали для преобразования в числовой тип функцию parseFloatQ, чтобы сохранить дробную часть числа. Когда массив чисел создан, можно применить к нему статистическую обработку с помощью, например, функции statisticQ, описанной в подразделе 1.7.2.

Теперь рассмотрим задачу считывания в массив значений столбца таблицы, управляемой элементом STD. Как известно, источником данных для такой таблицы является текстовый файл. В листинге 4.26 приводится пример HTML-документа с элементом управления STD, таблицей и сценарием, производящим чтение данных из столбца с одновременным переводом их в числовой тип, а также вычисляющим сумму всех значений.

Листинг 4.26. Пример HTML-документа с элементом управления STD

```
<HTML>
<! Элемент управления >
<OBJECT ID = 'mydatacontrol'
   CLASSID = 'CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83'>
   <PARAM NAME = 'FieldDelim' VALUE='|'>
   <PARAM NAME = 'DataURL' VALUE='mydata.txt'>
   <PARAM NAME = 'UseHeader' VALUE=true>
</OBJECT>

<! Таблица >
<TABLE ID="mytab">
<TR>
<TD><INPUT NAME = 'zp' DATASRC = #mydatacontrol
 DATAFLD = 'Зарплата' onchange = "zpchangeO">
</TD>
<! Здесь может быть определение других столбцов таблицы>
</TR>
</TABLE>
<P>
Сумма выплат <B ID = "sum"></B>

<SCRIPT>
```

```
var azp = new Array()
zpchange()                                  // вызов функции

function zpchange Q {
var 5 = 0, 1 = 0
mydatacontrol.recordset.moveFirst()         // переход к 1-й записи
while(!mydatacontrol.recordset.eof){
azp[i]  =  parseFloat(document.all.mytab.zp.value)
S+=  azp[i]
mydatacontrol.recordset.moveNext()          // переход к следующей записи
i++
}
document.all.sum.innerText = S              // вывод значения S
}
</SCRIPT>
</HTML>
```

Здесь предполагается, что источник данных находится в файле mydata.txt. В этом файле указано поле Зарплата, а поля разделены вертикальной чертой. Чтение данных поля Зарплата и вычисление суммы всех его значений происходит при загрузке документа в браузер, а также при каждом изменении в этом поле. Результат вычисления суммы S вставляется как текст в заранее подготовленный для этой цели элемент <B ID = "sum">.

4.8.10. Защита веб-страницс помощью пароля

Если мы хотим защитить сайт или его отдельную страницу с помощью пароля, то главное — обеспечить сохранность пароля, а также адреса защищаемой страницы. Очевидно, они не должны фигурировать в HTML-документе и сценарии в явном виде. Достаточно простой и надежный способ решения этой задачи — совпадение пароля с именем (без расширения) какого-нибудь файла, расположенного на сервере. В этом случае к слову, введенному пользователем, следует добавить расширение имени файла и проверить, существует ли он на сервере. Если файл существует, то введенный пользователем пароль верен, и можно открыть доступ к документу. В противном случае будет отказано в доступе. Обратите внимание, что имя файла в явном виде не упоминается. Проверку существования файла можно выполнить с помощью метода FileExistsQ объекта файловой системы (File SystemObject), который рассмотрен в главе 5. Однако использование этого объекта в сценарии для браузера сопряжено с появлением неприятного сообщения о том, что страница содержит элемент ActiveX, который может представлять опасность. При этом пользователю предлагается принять решение о выполнении программы. Если он откажется от ее выполнения, то не получит доступа к защищаемой странице.

Другой вариант решения задачи защиты с помощью пароля тоже основан на совпадении пароля с именем файла. Однако в этом варианте используется не объект файловой системы, а безопасный элемент ActiveX управления текстовыми базами данных. Файл с именем, совпадающим с паролем, является текстовым файлом, содержащим базу данных с двумя столбцами (полями) и двумя строками (записями). В первой строке записаны идентификаторы полей базы данных, между которыми указан символ-разделитель. Во второй строке через такой же разделитель тоже записаны всего лишь два слова — имя этого текстового файла

без расширения и имя HTML-файла или URL-адрес страницы перехода. Пусть поля базы данных имеют идентификаторы password и myspecurl Тогда содержимое текстового файла с именем, например, myspecfile.txt должно иметь следующий вид:

```
password]myspecurl
myspecfile|http://www.myweb.ru/mypage.htm
```

Здесь во второй строке во втором ее поле указан гипотетический URL-адрес страницы, на которую должен перейти браузер, если пользователь введет правильный пароль.

В следующем примере в HTML-документе размещается текстовое поле ввода и кнопка, щелчок на которой обрабатывается сценарием (листинг 4.27). Сценарий создает элемент управления и невидимую таблицу, загружая в нее базу данных. В этой таблице всего две ячейки: одна с паролем, а другая с адресом перехода. Далее слово, введенное пользователем, сравнивается со словом в таблице. Если они совпадают, то происходит переход на страницу с указанным адресом. В противном случае выдается сообщение о том, что введенный пароль неверен.

Листинг 4.27. Сценарий защиты HTML-документа с помощью пароля

```
<HTML>
<H3>Только для членов клуба</H3>
Введите пароль:
<INPUT ID = "pwl" TYPE="text" VALUE="">
<BUTTON ID = "pwenter">ВВОfl</BUTTON>
<B ID = "dbelem"></B>
<SCRIPT>
function pwenter.onclick() {
if (!document.all.pw1.value)          // если поле ввода пароля пусто
   return
dbstr = '<OBJECT ID = "mydbcontrol" ' +
   'CLASSID= "CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">' +
   '<PARAMNAME= "DataURL" VALUE = '" + document. all.pwl .value+ '.txt"> ' +
   '<PARAMNAME="FieldDelim" VALUE="|">'
   '<PARAMNAME = "UseHeader" VALUE = true></OBJECT>' +
   '<TABLE STYLE = "visibility:hidden"><TR><TD>' +
   '<INPUT ID = "pw2" TYPE = "text" DATASRC = "#mydbcontrol"' +
   'DATAFLD= "password"></TD><TD>' +
   '<INPUT ID = 'xurl" TYPE = "text" DATASRC = "tfmydbcontrol" ' +
   'DATAFLD = "myspecurl"x/TD></TRx/TABLE>'
document.all.dbelem.innerHTML=dbstr   /* вставляем элементы БД
                                         в документ */
setTimeout ("validationO", 1000)       // задержка
}

function validationO {                // проверка правильности и переход,
if (document.all.pwl.value == document.all.pw2.value){   /* если пароль
                                                            верен */
   document.all.pwl.value = ""
   window.location.href = document.all.xurl.value  // переход,
}else                                  // если пароль не верен
   alert("Пароль не верен!")

</SCRIPT>
</HTML>
```

Здесь использована временная задержка в 1 с, необходимая для того, чтобы браузер успел сделать вставку элементов с помощью свойства innerHTML до проверки пароля. Обратите внимание, что нигде в документе значение пароля, а также адрес страницы перехода не упоминаются в явном виде.

При желании рассмотренный выше сценарий можно усовершенствовать. Так, если введенный пользователем пароль оказался верным, то его можно сохранить в cookie-файле, с тем чтобы при следующем посещении страницы пользователю не пришлось снова его вводить. Об этом уже говорилось в разделе 2.9. Например, в загруженном в браузер документе имеется ссылка на защищенную паролем страницу. При щелчке на этой ссылке сценарий считывает из cookie-файла значение специальной записи, содержащей пароль. Далее в документ загружается соответствующий текстовый файл базы данных с паролем и производится проверка правильности пароля из cookie-записи. Если эта проверка оказалась успешной, то происходит переход по ссылке, указанной в базе данных (ссылка в документе не должна содержать адрес перехода в явном виде). В противном случае в текущем документе появляется поле и кнопка для ввода пароля с клавиатуры. Введенное пользователем слово сравнивается с паролем из базы данных. Если они совпали, то нужная страница загружается в браузер, а пароль сохраняется в cookie-записи. В противном случае выдается сообщение, что введенный пароль неверен.

В листинге 4.28 приведен пример HTML-документа со сценарием, который пытается взять пароль из cookie-записи с именем myspecrecord.

Листинг 4.28. Пример HTML-документа со сценарием, который пытается взять пароль из cookie-записи

```
<HTM1_>
<H3>Только для членов клуба</H3>
<a href="#"  ID ="myref"onclick="pwvalid()  ">Щелкни</a>
<B ID =  "dbelem"></B>
<SCRIPT>
var pw        /* глобальная переменная для полученного
                 из cookie или введенного пользователем пароля*/

function pwvalid(){                      // проверка пароля
pw = readCookie("myspecrecord")          // читаем cookie-запись
if  (!pw)                                /* нет нужной записи
                                            в cookie-файле */
   pw = pwinputO                         /* делаем поле ввода и кнопку,
                                            вызывающую pwenterQ */
else
   pwenter()                             /* загружаем БД, сравниваем и
                                            что-то делаем */
}

function pwinput(){                      // поле ввода и кнопка
var inpstr = 'Введите пароль: <INPUT ID = "pwl" TYPE = "text" VALUE = "">'+
   '<BUTTON onclick = "pw = document.all.pwl.value;  pwenter() ">Ввод
   </BUTTON>'
document.all.myref.outerHTHL = inpstr  // вставляем поле ввода и кнопку
}

function  pwenter(){
```

продолжение &

***Листинг 4.28** (продолжение)*

```
if  (!pw)                                    // если нет пароля
    return
dbstr = '<OBJECT ID = "mydbcontrol"  CLASSID =
  "CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">' + ,
  '<PARAMNAME = "DataURL" VALUE = "' + pw + '.txt"> ' +
  '<PARAMNAME="FieldOelim" VALUE="|">' +
  '<PARAM NAME = "UseHeader" VALUE = true></OBJECT><TABLE STYLE =
  "visibility:hidden"><TR><TD>'+
  '<INPUT ID = "pw2" TYPE = "text" DATASRC = "tfmydbcontrol" DATAFLD =
  "password"></TD><TD>' +
  '<INPUT ID = "xurl" TYPE = "text" DATASRC = "tfmydbcontrol" DATAFLD =
  "myspecurl"></TD></TR></TABLE>'
document.all.dbelem.innerHTML=dbstr     // вставляем элементы в документ
setTimeout("validation()",1000)         // задержка
}

function validation0 {                  // проверка правильности
                                        // и переход
if  (pw == document.all.pw2.value){     // если пароль верен,
   window.location.href = document.all.xurl.value    // переход
   var d= new Date()
   var d2=d.getTime()+(365*24*60*60*1000)
   d.setTime(d2)                        // срок хранения cookie-записи
   writeCookie("myspecrecord", pw, d)   // запись пароля в cookie-файл,
}else                                   // если пароль не верен
   alert("Пароль не верен!")
!!

function readCookie(name)  {            // чтение cookie-данных записи
var xname = name + "="
var xlen = xname.length
var clen = document.cookie.length
var i = 0
while(i < clen){
var j = i + xlen
if (document.cookie.substring(i , j) == xname)
      return getCookieVal(j)
i = document.cookie.indexOf(" ",!) + 1
if (i == 0)  break
}
return null
}

function getCookieVal(n){               /* вспомогательная функция,
                                           вызываемая из readCookie0 */
var endstr = document.cookie.indexOf(";", n)
if (endstr == -1)
   endstr = document.cookie.length
return unescape(document.cookie.substring(n, endstr))
}

function writeCookie(name , value, expires, path, domain, secure)  {
                                        /* запись cookie */
   document.cookie =
   name + "=" +escape(value) +
     ((expires) = "; expires=" + expires.toGMTString() : "") +
     ((path) ? "; path=" + path : "") +
```

```
    ((domain) ? "; domain=" + domain : "") +
    ((secure) ? "; secure" : "")
}
</SCRIPT>
</HTML>
```

Функция validation() в этом примере по сравнению с предыдущим несколько модифицирована: помимо проверки правильности пароля и перехода по заданному адресу она производит запись (перезапись) cookie. Срок хранения здесь устанавливается равным 1 году с момента текущей системной даты. Функции чтения readCookie() и записи writeCookie() данных в cookie-файл рассматривались в разделе 2.9. Далее в сценарии используется глобальная переменная pw для хранения пароля, либо полученного из cookie-записи, либо введенного пользователем. Эта переменная используется в функциях pwinput(), pwenter() и validation(). Я допускаю, что приведенный выше код не очень изящный (многовато функций), но он работает, поэтому я и решил не заниматься дальнейшими усовершенствованиями. Однако я рекомендую вам попытаться улучшить сценарий.

Тот факт, что имя файла (без расширения) базы данных и значение ее поля password совпадают со значением пароля, существенно усиливает защиту. Действительно, для доступа к защищенному документу необходимо не только существование на сервере файла с именем, совпадающим с паролем, но и чтобы этот файл содержал данные в определенной структуре. А именно он должен содержать свое имя-пароль. Очевидно, далеко не всякий текстовый файл обладает таким свойством.

С точки зрения идеологии парольной защиты веб-страниц сама собой напрашивается следующая модернизация рассмотренных выше сценариев. Пусть пароль совпадает с именем файла текстовой базы данных, но сама база содержит несколько записей. В первом столбце базы данных записан не пароль, а условные имена пользователей (Login). На загруженной в браузер веб-странице пользователь вводит свое имя (login) и пароль. Далее в текущий документ загружается невидимая база данных из текстового файла с именем, совпадающим с введенным паролем. В этой базе производится поиск записи, соответствующей имени пользователя. В случае успеха поиска в браузер загружается документ с адресом, указанным во втором поле. Таким образом на сайте можно хранить, например, индивидуальные сообщения для пользователей вашего клуба.

Заметим, что текстовый файл с базой данных паролей создается и поддерживается автором (владельцем) сайта. Он размещается на сервере вместе с другими файлами сайта. Пароль передается посетителям по электронной почте или телефону.

4.9. Взаимодействие с Flash-мультфильмами

Приложения, созданные в системе Macromedia Flash версий 5.0 и 6.0 (MX), могут взаимодействовать со сценариями на JavaScript. А именно они могут получать данные из сценария, написанного на языке JavaScript, и использовать их некоторым образом с помощью своего сценария, написанного на языке ActionScript. С другой стороны, сценарии на ActionScript могут использовать сценарии (функции) на JavaScript. Чтобы организовать такое взаимодействие, необходимо преж-

де всего вставить приложение Flash (мультфильм, ролик, клип) в HTML-документ. Это делается с помощью контейнерного тега <OBJECT>, инициирующего Flash-проигрыватель и загружающего в него мультфильм. В IE5+ этот проигрыватель уже встроен.

4.9.1. Передача данных из JavaScript в ActionScript

В качестве примера передачи данных из JavaScript в ActionScipt рассмотрим задачу отображения текста в окне Flash-мультфильма. Используя большие изобразительные возможности Flash, можно создать окно произвольной формы, а не только прямоугольное, как в HTML. В нашем примере HTML-документ содержит Flash-проигрыватель с загруженным в него мультфильмом, поле ввода данных и сценарий, который передает во Flash-мультфильм содержимое поле ввода. При этом содержимое поля ввода данных (элемента <INPUT>) отображается в окне Flash-мультфильма (рис. 4.34). В листинге 4.29 приведен соответствующий HTML-код.

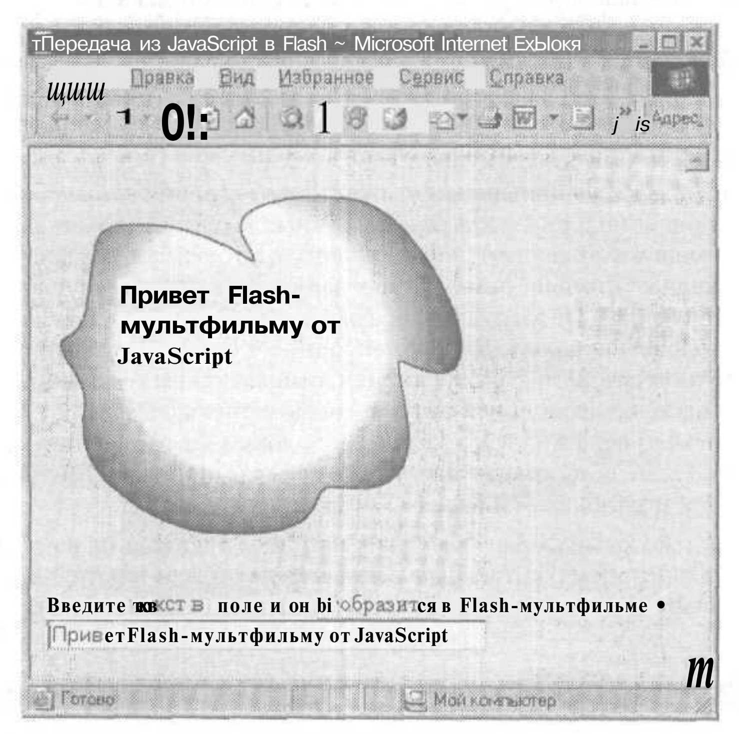


Рис. 4.34. Текст в поле ввода (<INPUT>) оттображается в окне Flash-мультфильма

Листинг 4.29. Пример передачи данных из JavaScript в ActionScipt

```
<HTML>
<HEAD>
<meta http-equiv = Content-Type content = "text/html; charset =
  windows-1251">
```

```
<TтE>Передача из JavaScript в Flash</TITLE>
</HEAD>
<BODY BGCOLOR = "#e0e0e0">

<! Flash-проигрыватель >
<OBJECT CLASSID = "clsid: D27CDB6E-AE6D-llcf-96B8-444553540000"
  CODEBASE =
  "http://download.fflacromedia.com/pub/shockwave/cabs/flash/
  swflash.cab#version = 6,0,0,0"
  WIDTH = "287" HEIGHT = "261" id = "myflash" ALIGN="">
<PARAM NAME = movie VALUE = "myflash.swf">
<PARAM NAME = quality VALUE=high>
<PARAM NAME = wmode VALUE = transparent
<PARAM NAME = bgcolor VALUE=#FFFFFF>
<EMBED src = "myflash.swf" quality=high wmode=transparent bgcolor =
  #FFFFFF WIDTH = "287" HEIGHT = "261" NAME = "myflash" ALIGN = ""
  TYPE = "application/x-shockwave-flash"
  PLUGINSPAGE = "http://www.macromedia.com/go/getflashplayer">
</EMBED>
</OBJECT>
<P>
Введите текст в поле, и он отобразится во 1a5п-мульфильме<BK>
<INPUT TYPE = "text" NAME = "inputtext" SIZE = 40 onchange = getit()>
</BODY>

<SCRIPT>
function getit(){
document.myflash.SeW ariable("inFlash", inputtext.value)
}
</SCRIPT>
</HTML>
```

Здесь с помощью тега <OBJECT> в браузер загружается элемент управления ActiveX, соответствующий Flash-проигрывателю версии 6.0 (MX), хотя для воспроизведения нашего мультфильма достаточно и версии 5.0. В этот проигрыватель, в свою очередь, загружается мультфильм из файла myflash.swf. Атрибут CODEBASE тега <OBJECT> содержит в качестве значения URL-адрес проигрывателя, если он не встроен в браузер пользователя. Заметим, что в IE5+ Flash-проигрыватель уже встроен. Вложенный контейнерный тег <EMBED> используется только ради браузера Netscape Navigator. Далее в HTML-документе с помощью тега <INPUT> задается поле ввода данных. При изменении значения этого поля (событие onchange) вызывается функция getit(), определенная в сценарии JavaScript. В этой функции с помощью метода SetVariable() переменной inFlash присваивается значение поля ввода данных inputtext.value. Этого вполне достаточно, чтобы передать значение поля ввода данных в мультфильм. Метод SetVariable() является методом объекта Flash-проигрывателя, который в данном примере имеет ID = "myflash". Имя переменной inFlash (произвольное) задается как один из параметров Flash-мультфильма. Среди прочих параметров мультфильма отметим еще <PARAM NAME = wmode VALUE = transparent:»., который задает прозрачность фона занимаемой им прямоугольной области. Это простой и весьма эффективный способ согласования содержимого Flash-мультфильма с общим дизайном веб-страницы.

Теперь рассмотрим, как сделать Flash-мультфильм, воспринимающий и отображающий данные из HTML-документас помощью сценария, написанного наязыке

JavaScript. В пакете Macromedia Flash MX создадим простой мультфильм, содержащий два слоя. В первом, фоновом слое нарисуем окно. Это может быть любая фигура. В нашем примере я использовал инструмент Овал, чтобы нарисовать эллипс. Затем с помощью инструмента Subselection (белая стрелка) я преобразовал эллипс в некую причудливую форму, напоминающую облако. Более того, я залил ее радиальным градиентом. Далее необходимо создать второй слой, расположенный над первым. В этом слое в пределах «облака», с помощью инструмента Text создается текстовое поле, параметры которого определяются в палитре Properties (Свойства). Эта палитра во Flash MX еще называется Инспектором свойств. Сначала зададим тип текстового поля — Input Text (Ввод текста). Затем в поле Var (Переменная) введем имя переменной, принимающей текст, который должен отображаться в текстовом поле. В рассматриваемом примере я выбрал имя переменной inFlash. Это важный момент, поскольку именно эта переменная указывается в сценарии JavaScript как первый параметр метода SetVariable(). Наконец, с помощью других органов управления палитры Properties задаем дополнительные параметры текстового поля (рис. 4.35). В частности, выбираем из раскрывающегося списка имя шрифта, режим отображения текста Multiline (Многострочный) и др.

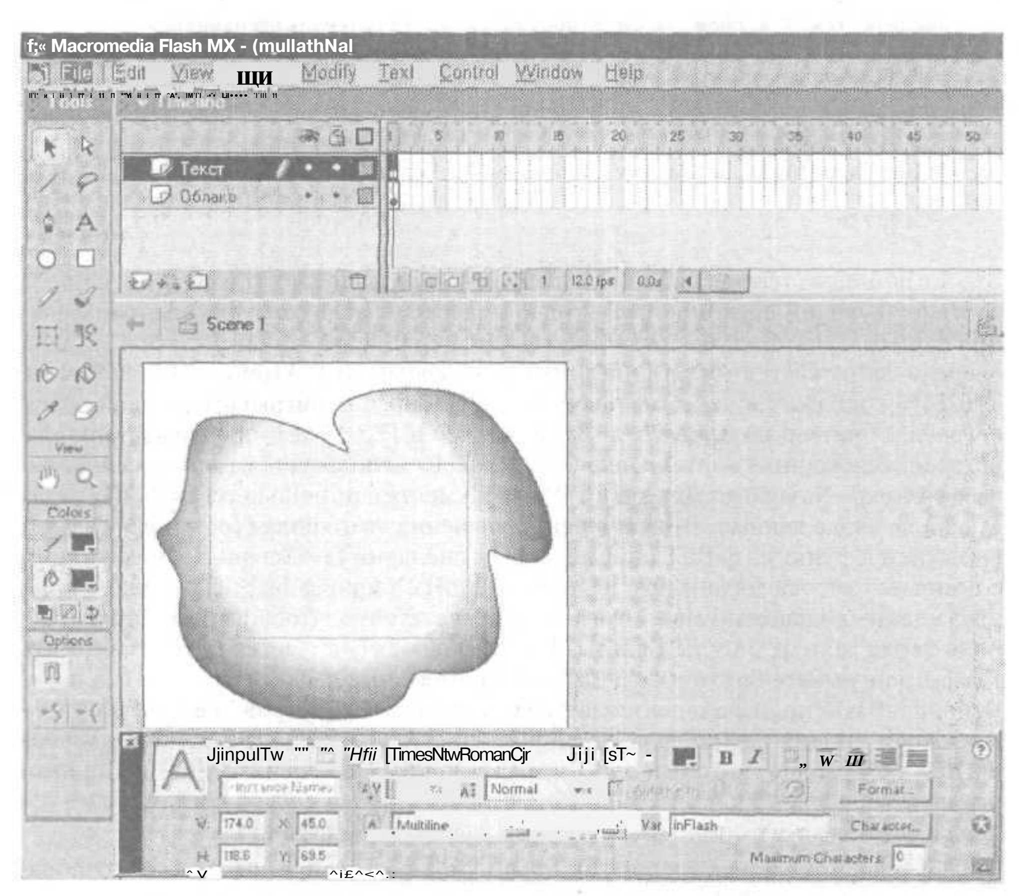


Рис. 4.35. Создание Flash-мультфильма, отображающего текст из HTML-документа с помощью сценария на JavaScript. Внизу — палитра Properties

Исходный файл созданного мультфильма сохраним под именем myflash.fla. Это необходимо, если нам потребуется в дальнейшем модифицировать его. Для вставки мультфильма в HTML-документ необходимо создать файл в формате SWF, то есть файл с расширением .swf. Этой цели служит так называемая операция публикации мультфильма Publish, которая выбирается из меню File (Файл) в главном окне пакета Flash. Параметры публикации можно задать по команде File У Publish Settings (Файл ▸ Настройка публикации). В соответствующем диалоговом окне можно задать имя swf-файла, версию Flash-проигрывателя и другие параметры. Вы можете также потребовать создание HTML-файла, который содержит фрагмент HTML-кода (элемент <OBJECT>), загружающий Flash-проигрыватель и мультфильм со всеми параметрами, указанными в диалоговом окне настроек. Этот фрагмент можно просто скопировать в создаваемый HTML-документ вашего проекта.

4.9.2. Вызов сценария JavaScript из сценария ActionScript

Язык ActionScript для создания сценариев Flash-мультфильмов — достаточно мощное средство программирования, поэтому он может обойтись и без JavaScript. Тем не менее существует возможность вызывать фрагменты JavaScript-кода из сценария, написанного на ActionScript. Это удобно в тех случаях, когда у вас уже созданы программы на JavaScript. Например, зачем писать код функции представления чисел словами на ActionScript, если он уже написан (см. раздел 4.4) на JavaScript?

Вызов функции JavaScript из сценария ActionScript осуществляется следующим образом:

```
getURL("javascript: имя_функции(параметры)")
```

В нашем примере Flash-мультфильм содержит три кнопки, щелчки на которых обрабатываются функциями, написанными на JavaScript. Если уж быть точным, то щелчок на кнопке в мультфильме обрабатывается сценарием на ActionScript, который вызывает функцию на JavaScript.

HTML-документ содержит тег <OBJECT>, загружающий Flash-проигрыватель вместе с мультфильмом, и сценарий, в котором определены три функции: открытия нового окна, вывода диалогового окна с сообщением и закрытия окна. В листинге 4.30 приведен соответствующий код.

Листинг 4.30. Вызов сценария JavaScript из сценария ActionScript

```
<HTML>
<HEAD>
<meta http-equiv = Content-Type content = "text/html;
  charset = windows-1251">
<TITLE>Вызов JavaScript из ActionScript</TITLE>
</HEAD>
<BODY BGCOLOR = "#e0e0e0">
<! Flash-проигрыватель >
<OBJECT CLASSID = "clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
  CODEBASE = "http://download.macromedia.com/pub/shoekwave/cabs/flash/
  swflash.cab#version = 6,6,0,0" WIDTH = "400" HEIGHT = "100" id =
  "myflash" ALIGN= "">
```

продолжение ⇨

Листинг 4.30 *(продолжение)*

```
<PARAM МАЙЕ = movie VALUE = "myflash.swf">
<PARAM NAME = quality VALUE=high>
<PARAM NAME = wmode VALUE = transparent^
<PARAM NAME = bgcolor VALUE=#FFFFFF>
<EMBED src = "myflash.swf" quality=high wmode=transparent bgcolor =
  #FFFFFF WIDTH = "287" HEIGHT = "261" NAME = "myflash" ALIGN = ""
  TYPE = "application/x-shockwave-flash" PLUGINSPAGE =
  "http://www.macromedia.com/go/getflashplayer">
</EMBED>
</OBJECT>
</BODY>
<SCRIPT>
var mywindow
function newwindow(myURL, mywidth, myheight){ /* открытие нового окна */
mywindow = wi ndow.open(myURL,'saple' , 'saple"toolbar=no,ar= bar=no, location=no,
  menubar=no, scrollbars=no, resizable=no, width=" + mywidth +
  ", height=" + myheight + ", top=0, left=0")
I
function message(message){                          // вывод сообщения
alert(message)
I
function closewindowO{                              // закрытие окна
mywindow
mywindow.close()
}
</SCRIPT>
</HTML>
```

Во Flash-редакторе создадим простой мультфильм, содержащий три кнопки. Для этого можно воспользоваться встроенной библиотекой, содержащей множество уже готовых элементов управления. Каждой кнопке назначим следующие действия (сценарии):

Кнопка 1 (открытие окнаизагрузкавнегофайлаtemp.htm):

```
on(release){
getURL("javascript: newwindow('temp.htm' , 550, 250)")
}
```

Кнопка 2 (вывод сообщения):

```
on(release){
getURL("javascript: message('Привет из Flash')")
}
```

Кнопка 3 (закрытие окна):

```
on(release){
getURL("javascript: closewindowO")
I
```

Исходный файл созданного мультфильма сохраним под именем **myflash,fla.** Это необходимо, если нам потребуется в дальнейшем модифицировать его. Для вставки мультфильма в HTML-документ необходимо создать файл в формате SWF, то есть файл **myf lash.swf.** Этой цели служит операция публикации мультфильма Publish, которая выбирается из меню File (Файл) в главном окне пакета Flash. Параметры публикации можно задать по команде File > Publish Settings (Файл > Настройка публикации). В соответствующем диалоговом окне можно задать имя **swf-файла,** версию Flash-проигрывателя и другие параметры.

Глава 5. Работа с файловой системой и реестром Windows

Операции с дисками, папками и файлами с помощью программ являются очень важными для программистов, поскольку серьезные проекты невозможны без взаимодействия с файловой системой (создание, перемещение, удаление папок и файлов). В JavaScript эти операции имеют некоторые особенности по сравнению с другими языками.

Доступ к файловой системе с помощью языков на основе сценариев, таких как JavaScript и VBScript, в Windows обеспечивается через объект FileSystemObject (FSO — объект файловой системы). Программы на JavaScript и VBScript, использующие этот объект, могут интерпретироваться браузером IE5+, а также системой Windows Scripting Host (WSH), встроенной в Windows 98 SE и более поздние версии (ее также называют Windows Based Script Host).

Операциям с файловой системой, выполняемым браузером пользователя с помощью сценариев, даже при установленном самом низком уровне безопасности будут предшествовать предупреждающие сообщения о возможных неприятностях. Предупреждения будут появляться и при работе на локальном .компьютере без использования сети. Это сделано для того, чтобы недобросовестные посетители Интернета не могли нанести вред компьютеру пользователя. Именно поэтому рекомендуется использовать FSO не на клиентском компьютере, а на сервере (технология Active Server Pages, ASP). WSH, в отличие от браузера, позволяет свободно использовать FSO на локальном компьютере: вы просто создаете программу на JavaScript в текстовом файле с расширением js и выполняете ее с помощью так называемого сервера сценариев Windows (файл wscript.exe, расположенный в папке Windows). Эта программа устанавливается по умолчанию как приложение для открытия (выполнения) файлов с расширениемjs. Аналогичная программа для запуска сценариев через командную строку MS DOS представлена файлом cscript.exe.

Следует иметь в виду, что некоторые антивирусные пакеты (например, Norton AntiVirus) позволяют и рекомендуют устанавливать блокировку сценариев. Если блокировка установлена, то, в зависимости от настройки, сценарий не будет выполняться либо будет выводиться окно с предупреждением и предложением выбора возможных вариантов действий (например, запретить выполнение, разрешить выполнение один раз или всегда и др.).

Программы JavaScript, написанные для выполнения браузером и WSH, во многом похожи. Однако имеются и различия. Так, сценарии для браузера размеща-

ются в HTML-документе (обычно в контейнере <SCRIPT>) или в js-файле. Программы для WSH размещаются только bjs-файлах. Для вывода сообщений в браузере используется, например, метод alertQ. В WSH такого метода нет. Вместо него там применяется метод WScript.EchoQ, отсутствующий в браузере. Есть и другие различия.

При работе с файловой системой и реестром Windows следует соблюдать осторожность, поскольку можно нечаянно не только потерять ценные данные, но и повредить операционную систему.

5.1. Создание объекта файловой системы

Итак, чтобы получить доступ к файловой системе, необходимо создать для нее объект FileSystemObject (точнее говоря, экземпляр FSO). Если ваша программа на JavaScript будет выполняться браузером как сценарий в HTML-документе, то для создания FSO можно использовать только следующее выражение:

```
var fso = new ActiveXObject("Scripting.FileSystemObject)
```

Если программа предназначена для выполнения с помощью WSH, то кроме указанного выше выражения можно использовать еще и такое:

```
var fso = WScript.CreateObject("Scripting.FileSystemObject")
```

Здесь fso — переменная (ее имя может быть произвольным), содержащая ссылку на объект файловой системы. Эта ссылка будет использоваться для применения методов и свойств объекта файловой системы. В дальнейшем мы будем применять первый вариант создания FSO, поскольку он подходит и для браузера, и для WSH. Заметим, что первый вариант соответствует вызову объекта FSO как элемента управления ActiveX, а второй — как объекта приложения Wscript. Но на этих деталях мы не будем здесь останавливаться.

После того как объект файловой системы создан, можно применить методы для создания и удаления папок и файлов, копирования и перемещения файлов, а также получения информации о дисках, папках и файлах. Существуют и другие методы, такие как открытие и закрытие файла, запись данных в файл и т. п. Мы рассмотрим их в следующих разделах данной главы. Общий же синтаксис таков:

Ссылка на объект файловой системы (FSO) для доступа к ее объектам:

```
var fso = new ActiveXObject("Scripting.FileSystemObject")
```

Методы доступа к существующим объектам и методы создания новых объектов:

```
var x = fso.методОбъекта(параметры)
```

Свойства и методы конкретного объекта — диска, папки или файла:

```
var y = x.свойство(параметры)
var z = x.метод(параметры)
```

Например, для получения информации о свободном пространстве на диске С следует выполнить следующий код:

```
var fso = ActiveXObject("Scripting.FileSystemObject" ) // ссылка на FSO
ar d = fso.GetDrive("C")     /* ссылка на объект
                                с характеристиками диска С */
var freespace = d.FreeSpace // значение свободного пространства в байтах
Wscript.Echo(freespace)     // вывод сообщения
```

Подробности будут изложены в следующих разделах данной главы. При этом обратите внимание, что ряд методов FSO разбивается на две группы: Get-методы и Create-методы. Названия методов из той или иной группы начинаются либо с Get, либо с Create. Get-методы предназначены для получения ссылок на уже существующие объекты (get — получить). Create-методы предназначены для создания объектов (create — создать). Однако методы создания возвращают ссылку на созданный объект. Поэтому если вы создали объект Create-методом, а затем вам нужна ссылка на него, то лучше сохраните ссылку, возвращаемую Create-методом в переменной для дальнейшего использования, а не применяйте Get-метод. Окончательно понять смысл всех этих наставлений вы сможете, когда столкнетесь с написанием программы, в которой необходимо то создавать объекты файловой системы, то получать информацию о них. Общее правило следующее: Get-методы работают для уже существующих объектов, а Create-методы, кроме создания новых объектов, обеспечивают вам и доступ к этим новым объектам, такой же как и Get-методы.

5.2. Работа с дисками

Работа с дисками заключается в получении информации о них (объем свободного пространства, тип, готовность и т. п.). Мы не можем, разумеется, ни создавать, ни удалять диски. Информация о дисках важна, например, при создании папок и файлов. При создании, копировании или перемещении файла полезно сначала убедиться, что указанный вами диск существует, готов к работе и имеет достаточно свободного пространства. Знание серийного номера диска может использоваться, например, при решении задачи защиты программных продуктов от несанкционированного копирования.

Сначала создается объект FSO для доступа к файловой системе, и ссылка на него сохраняется в переменной, например fso. Далее используются методы и свойства для получения информации о диске. Пусть переменная dpath содержит букву, которой обозначен диск, или путь к какой-нибудь папке, начинающийся с буквы диска. Тогда метод fso.GetDrive(path) возвращает ссылку на объект, содержащий информацию о диске, указанном в dpath. Пусть эта ссылка сохранена в переменной d. Теперь остается только получить значения свойств этого объекта, которые являются характеристиками диска. Например, свойство d.IsReady равно true, если диск готов, и false — в противном случае. Свойство d.FreeSpace содержит величину свободного пространства на диске в байтах. Чтобы определить, существует ли указанный в dpath диск, необходимо применить метод fso.DriveExists(dpath). Этот метод возвращает 0, если диск не существует, в противном случае — 1.

Ниже приведен код функции driveinfo(dpath), которая возвращает массив всех характеристик диска, указанного в качестве строкового параметра:

```
function driveInfo(dpath) {              // информация о диске
// Возвращает массив характеристик диска
var fso = ActiveXObject("Scripting.FileSystemObject")
var disk = fso.GetDriveName(dpath)       // имя (буква) диска
var dinfo = new Array(7)
if (fso.DriveExists(disk)){              // если диск существует
```

```
    var d = fso.GetDrive(disk)           // объект с информацией о диске
    dinfo[0] = d.DriveLetter             // буква диска
    dinfo[1] = d.IsReady                 // готовность диска
    dinfo[2] =d.DriveType                // тип диска
    if (d.IsReady) {
      dinfo[3] = d.VolumeName            // имя диска
      dinfo[4] = d . SerialNumber        // серийный номер диска
      dinfo[5] = d.TotalSize             // полный объем в байтах
      dinfo[6] = d.FreeSpace             // свободно в байтах
    }
  }
  return dinfo                /* возвращение массива характеристик
                                 диска */
}
```

Обратите внимание, что некоторые характеристики диска мы получаем только после проверки готовности диска (например ,гибкий диск установлен в дисковод). Мы используем выражение var disk = fso.GetDriveName(dpath) на тот случай, когда параметр функции содержит не просто букву диска, а путь к папке или файлу.

Чтобы протестировать функцию driveinfo(dpath), напишите следующий код:

```
function driveInfo(dpath) {
  . . .                          // код функции
}

WScript.Echo(driveInfo("A"))
WScript.Echo(driveInfo("C"))
WScript.Echo(driveInfo("D") )
WScript.Echo(driveInfo("E"))
WScript.Echo(driveInfo("C: \\Моидокументы") )
var x = drive!nfo("C")
WScript.Echo( "Свободно: " +x[5])
```

Сохраните этот код в файле с расширениемjs и выполните его (дважды щелкните на этом файлев Проводнике).

Функция driveTotalInfo(dpath) (пример см. на рис. 5.1), код которой приведен в листинге 5.1, ничего не возвращает, а выводит диалоговое окно с характеристиками одного или всех дисков. Так, в качестве строкового параметра можно указать один интересующий нас диск. Однако если параметр не указан или пуст, выводится информация обо всех дисках.

Листинг 5.1. Код функции driveTotalInfo(dpath)

```
function driveTotaUnfo(dpath) {            // Информация о дисках
var fso = new ActiveXObject("Scripting.FileSystemObject")
var di sk
if (!dpath)                                /* если нет параметра,
                                              то все диски */
disk = new Array ( "A", "B" , "C" , "D" , "E" , "F" , . . . "Z")
else
disk = new Array (fso.GetDriveName(dpath) )   // одноэлементный массив
var s = "", t, d
ford = 0; i < disk.length: i++){
if (fso.DriveExi sts(disk [i ] ) ) {           // если диск существует
   d = fso.GetDrive(disk[i] )                 // ссылка на диск
   switch (d . DriveType) {                   // тип диска
      case 0: t = " - неизвестный"; break
```

```
            case 1: t = " - съемный"; break
            case 2: t = " - несъемный"; break
            case 3: t = " - сетевой"; break
            case 4: t = " - CD-ROM"; break
            case 5: t = " - виртуальный"; break
        }
        s+= "Диск " + d. DriveLetter + ":"            // буква диска

        if  (d.IsReady){                             // если диск готов
            s+= d .VolumeName + t                    // имя диска
            s+= "\n SN: " + d.SerialNumber           // серийный номер диска
            s+= "\n Объем: '" + d.TotalSize          // полный объем в байтах
            s+= "\n Свободно: " + d.FreeSpace        // свободно в байтах
        }else
            s+= t+ '" не готов"
        s+= "\n\n"
    }
    ;
    WScript.Echo(s)                                  /* вывод строки
                                                        с характеристиками */
    }
```

Чтобы использовать функцию driveTotallnfo(dpath) в сценарии, выполняем'ом браузером, необходимо лишь заменить в ней выражение WScript.Echo(s) haalert(s).

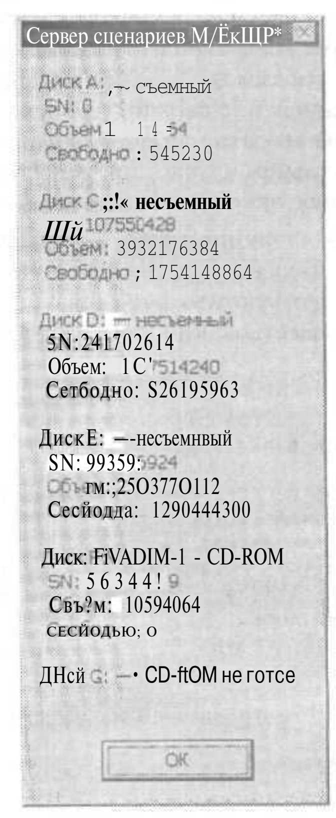


Рис. 5.1. Пример результата работы функции driveTotal!nfo()

5.3. Работа с папками

В этом разделе мы научимся создавать папки и работать с ними.

5.3.1. Создание папки

Для создания папки можно использовать следующий код:

```
var fso = new ActiveXObject("Scripting.FileSystemObject")
fso.CreateFolder(folderpath)
```

Здесь folderpath — строка, содержащая полный путь к создаваемой папке, например "C:\\Мои документы\\моя папка". Обратите внимание, что при указании пути требуется использовать двойные слэши. Напомним, что для выполнения с помощью WSH вместо первого выражения можно использовать и такое:

```
var fso = wscript.CreateObject("Scripting.FileSystemObject")
```

Второе выражение, fso.CreateFolder(fotderpath), создает указанную папку и возвращает ссылку на нее, если операция прошла успешно. В противном случае выводится диалоговое окно с сообщением об ошибке.

Если требуется создать папку, то необходимо, чтобы существовали все папки более высокого уровня, лежащие на пути к создаваемой папке и указанные в параметре folderpath. Кроме того, диск, на котором создается папка, должен быть записываемым и готовым к работе. Например, это не может быть устройство для чтения компакт-дисков, а если это гибкий диск, то он должен быть вставлен в дисковод. Наконец, если папка, указанная в folderpath, уже существует, то ее нельзя создавать. Таким образом, чтобы создать папку, предварительно следует выполнить целый ряд проверок. В противном случае, при невыполнении оговоренных выше условий, появится диалоговое окно с сообщением об ошибке.

В листинге 5.2 приведен код функции createFolder(folderpath), которая выполняет все необходимые проверки. Более того, если какие-нибудь папки на пути к конечной (целевой) папке не существуют, то функция создаст их. Таким образом, вам не придется заботиться о существовании папок вышестоящего уровня.

Листинг 5.2. Код функции createFolder(folderpath)

```
function createFolder(folderpath){            // создание папки
/* Возвращает: -1. если папка создана или существует, и в - в поотивном
   случае */
var fso = new ActiveXObject("Scripting.FileSystemObject")
var disk = fso.GetDriveName(folderpath)       // имя (буква) диска
/* Проверка характеристик диска: */
if (!fso.DriveExists(disk)) return 0          // если диск не существует
if (!fso.GetDrive(disk).IsReady) return 0     // если диск не готов
// Если не подходит тип диска:
if (fso.GetDrive(disk).DriveType == 0 || fso.GetDrive(disk).DriveType ==
4)
return 0
if (fso.GetDrive(disk).FreeSpace< 1024) return 0   // если мало места
if (fso.FolderExists(folderpath)) return -1   /* если папка уже
                                                  существует, то не создаем ее */
```

```
var apath = folderpath.split("\S")("\\")   // преобразуем в массив имен папок
for(i = 1; i < apath.length; i++){
   disk+= "\\" + apath[i]
   if <!fso.FolderExists(disk))        // если папка не существует,
      fso.CreateFolder(disk)           // то создаем ее
>
return fso.FolderExists(folderpath)  /* возвращаем результат проверки
                                         существования созданной папки */
}
```

Здесь дополнительно проверяется наличие свободного пространства на диске. Мы требуем, чтобы диск имел минимум 1 Кбайт свободного места, чтобы разрешить создание папки. Вы можете задать и другую пороговую величину или отменить эту проверку совсем.

5.3.2. Копирование, перемещение и удаление папки

Для копирования, перемещения и удаления папки используются следующие методы объекта файловой системы.

- CopyFolder(foLderpathl, folderpath2! [, переписать]) — копирует папку, указанную в строке folderpathl, в папку, указанную в строке folderpath2; если третий необязательный параметр имеет значение true, то уже существующая папка folderpathZ с тем же именем переписывается.
- MoveFolder(folderpathl, folderpathZ)) — перемещает папку, указанную в строке folderpathl, в папку, указанную в строке **folderpathZ.**
- DeleteFolder(folderpath [,force])) — удаляет папку, указанную в строкеfolderpath; если второй необязательный параметр имеет значение true, то удаляется и папка, предназначенная только для чтения.

Примеры

```
var folderpathl = "C:\\Мои доукменты\\Te511"
var folderpath2 = "C:\\Test2"
var fso=new ActiveXObject("Scripting.FileSystemObject")     // объект FSO
/* Создаем папку C:\Test2\ Мои доукменты\Tез11 */
fso.CopyFolder(folderpath1, folderpath2)
/* Удаляем папку C:\Test2 */
fso.DeleteFolder(folderpath2)
/* Создаем папку C:\Program Files\ Мои доукментыXTeз!! */
fso.MoveFolder(folderpath1, "C:\\Program Files")
```

Так же, как и в случае создания папки, при практическом копировании, перемещении или удалении папки необходимо сначала убедиться в том, что это действительно можно сделать. Следующая **функция** выполняет ряд проверок и в случае положительного результата удаляет папку:

```
function deleteFolder(folderpath){              // удаление папки
/* Возвращает
   0, если папка удалена или не существует, и
   -1 - в противном случае */
var fso = new ActiveXObject("Scripting.FileSystemObject")
if (!fso.FolderExists(folderpath)) return 0   // если папка не
                                              // существует
var disk = fso.GetDriveName(folderpath)        // имя (буква) диска
// Если не подходит тип диска:
```

```
if (fso.GetDrive(disk).DriveType == 0 || fso.GetDrive(disk).DriveType ==
4)
return -1
fso.DeleteFolder(folderpath)          // удаляем папку
return fso.FolderExists(folderpath)   /* возвращаем результат проверки
                                         существования созданной папки */
}
```

Напишите в качестве упражнения аналогичные функции для копирования и перемещения папок.

5.4. Работа с файлами

Вы можете создавать, удалять и перемещать текстовые файлы, а также читать их содержимое и характеристики, записывать в них данные.

5.4.1. Создание текстового файла

Чтобы создать текстовый файл на диске, следует выполнить следующие выражения:

```
var fso = new ActiveXObject("Scripting.FileSystemObject")
fso. CreatelextFUe(fUepath)
```

Здесь filepath — строка, содержащая полный путь к создаваемому файлу, например " C:\\M о и flOKyMeHTbi\\testfile. txt". Обратите внимание, что при указании пути требуется использовать двойные слэши. Напомним, что для выполнения с помощью WSH вместо первого выражения можно использовать и такое:

```
var fso = WScript.CreateObject("Scripting.FileSystemObject")
```

Второе выражение, **fso.CreateTextFile(filepath),** создает указанный файл, открывает с доступом для записи и возвращает ссылку на него, если операция прошла успешно. В противном случае выводится диалоговое окно с сообщением об ошибке. Обратите внимание, что созданный файл остается не доступным для записи.

Второй способ создания текстового файла основан на применении метода OpenTextFile (открыть текстовый файл) с параметром режима открытия **ForWriting** (для записи). Этот параметр имеет значение 2. Это делается следующим образом:

```
var fso = new ActiveXObject("Scripting.FileSystemObject")
var f = fso.OpenTextFile(fHepath, 2)
```

Новый текстовый файл, созданный описанными выше методами, ничего не содержит. Наполнение его данными производится специальными методами, которые будут рассмотрены в подразделе 5.4.3. Там же вы познакомитесь с методами чтения данных из уже существующего текстового файла. Обратите внимание, что создаваемый и открываемый для чтения или записи текстовый файл совсем не обязательно должен иметь расширение txt. Он может иметь расширение htm, html, shtml, js, asp, prg или др. Главное, чтобы он был текстовым по своему типу.

На практике при создании файла на диске прежде всего необходимо убедиться в возможности это сделать, чтобы избежать появления сообщений об ошибках. Так, если хотя бы одна из папок в пути к файлу не существует, то попытка применить метод CreateTextFileQ приведет к ошибке. Ошибка возникнет и в случае него-

товности диска, отсутствия указанного дисковода, а также в случае, если это устройство для чтения компакт-дисков. В листинге 5.3 приведен код функции createFue(filepath), которая выполняет все необходимые проверки. Кроме того, создаются папки, указанные в filepath, но не существующие.

Листинг 5.3. Код функции createFile(filepath)

```
function createFile(filepath){                // Создание текстового файла
/* Возвращает ссылку на созданный файл или 0, если файл не создан */
var fso = new ActiveXObject("Scripting.FileSystemObject")
var i = filepath.lastIndexOf("\\")
if (i >= 0) file = filepath.substr(i + 1)   /* выделяем имя файла из
                                                 filepath */
var folder = filepath . slice(0,i)         /* выделяем путь к файлу
                                              без имени файла */
if (!createFolder(folder)) return 0         /* проверки и создание
                                              недостающих папок */
if (fso.FileExists(file))                   // если файл существует, то
   return fso.OpenTextFile(filepath , 2)    // открываем его для записи

   return fso.CreateTextFile(folder + "\\" + file)   /* создаем файл и
                                                        возвращаем ссылку
                                                        на него */
```

Здесь используется функция createFolderQ создания папки, которая описана в подразделе 5.3.1. Она производит все проверки и при необходимости создает недостающие папки. Обратите внимание, что если указанный в параметре filepath файл уже существует на диске, то он открывается для записи.

В рассмотренной выше функции createFHe(filepath) для выделения имени файла из его полного имени filepath мы использовали встроенные функции JavaScript: lastIndexOfQ, substrQ и slice(). Однако вместо этого можно было использовать и специальные методы объекта файловой системы:

- GetBaseName(filepath) — возвращает последний элемент в filepath без расширения;
- GetExtensionName(filepath) — возвращает расширение последнего элемента в filepath.

Примеры

```
var fso = new ActiveXObject("Scripting.FileSystemObject")
   fso.GetBaseName("C:\\Иои документы\\testfile.txt")          //.testfile
   fso. GetExtensionName("C: \\Иои flOKyivieHTbi\\testf i le. txt")   // txt
```

Не менее полезным является и метод BuildPath(path, name), который к пути path дописывает элемент name:

```
var fso = new ActiveXObject("Scripting.FileSystemObject")
fso.BuildPath("C:\\ Мои документы", "testfile.txt")
```

Выше уже отмечалось, что если файл был создан, то он остается открытым. Чтобы закрыть файл, используется метод CloseQ.

Примеры

```
// Создание и закрытие файла:
var fso = new ActiveXObject("Scripting.FileSystemObject")
var myfile = fso.CreateTextFile("C:\\Мои документы\\testfile.txt")
```

```
var myfile.Close()

// Открытие и закрытие файла:
var fso = new ActiveXObject("Scripting.FileSystemObject")
var f = fso.OpenTextFile("C:\\Мои документы\\testfile.txt ", 2)
var myfile.Close0

// Создание/открытие и закрытие файла:
var myfile = createFile("C : \\Мои документы\\1ез1Т i le . txt ")
var myfile .Close()
```

5.4.2. Копирование, перемещение и удаление файла

Для операций копирования, перемещения (переименования) и удаления файлов имеются методы объекта файловой системы (FSO) и методы объекта файла.

```
var fso = new ActiveXObject("Scripting.FileSystemObject") // объект FSO
var file = fso.GetFile(filepathl)                          // объект файла
/* Методы копирования filepathl в filepath2 */
file.Copy(filepath2)
fso.CopyFile(filepathl.filepath2)

/* Методы перемещения filepathl в filepath2 */
file.Move(filepath2)
fso.MoveFile(filepathl.filepath2)

/* Методы удаления filepathl */
file.Delete(filepathl)
fso.DeleteFile(filepathl)
```

Также как и в случае с папками, методы копирования и удаления имеют еще один необязательный параметр. Так, значение true этого параметра в методах копирования обеспечивает перезапись уже существующего файла с тем же именем, а в методах удаления — удаление файлов, предназначенных только для чтения. Перечисленные выше операции могут применяться к любым файлам, а не только к текстовым.

В следующем примере создается текстовый файл testfile.txt в папке C:\Мои документы, затем этот файл перемещается в папку C:\Windows\Temp и копируется в корневую папку на диске C. В заключение он удаляется из обеих папок.

```
var fso = new ActiveXObject("Scripting.FileSystemObject")
var f1 = fso.CreateTextFile("C:\\Мои документы\\testfile.txt")
f1.Close()                                              // закрываем файл
fl = fso.GetFile("C:\\Мои документы\\testfile.txt") /* Получаем ссылку
                                                          на файл */
f1.Move("C:\\Windows\\Temp\\testfile.txt") /* Перемещаем файл в папку
                                              C:\Windows\Temp */
var f2=fso.GetFile("C:\\Windows\\Temp\\testfile.txt") // получаем ссылку
f2.Copy ("C:\\testfile.txt")                         /* копируем файл
                                                          в папку C:\ */
/* Получаем ссылки на файлы: */
fl = fso.GetFile("C:\\Windows\\Temp\\testfHe. txt")
f2 = fso.GetFile("C:\\testfile.txt")
f1.Delete()                                         // удаляем файл
C:\Windows\Temp\testfile.txt
f3.Delete()                                        /* удаляем файл
                                                      C:\testfile.txt */
```

Копировать, перемещать или удалять можно только закрытые файлы. Поскольку после операции создания файла он остается открытым, нам пришлось использовать метод **CloseQ.**

ВНИМАНИЕ —

Прежде чем копировать и перемещать файлы, необходимо проверить, возможны ли данные операции. Так, следует проверить, существует и готов ли диск, достаточно ли свободного места на нем, существуют ли все папки, указанные в пути к файлу. Требуется также решить, что делать, если копируемый или перемещаемый файл уже создан в месте назначения.

Некоторые из этих проверок следует выполнять и перед удалением файла. Попробуйте в качестве упражнения написать код функции, выполняющий все эти операции. Отчасти эта задача аналогична созданию и удалению папки (см. раздел 5.3). Для ее решения вам дополнительно потребуется информация о такой характеристике файла, как его объем.

Значение объема (размера) файла в байтах содержится в свойстве Size объекта файла. В следующем примере мы узнаем объем файла **C:\autoexec.bat:**

```
var fso = new ActiveXObject("Scripting.FileSystemObject")
var fl = fso.GetFile("C:\\autoexec.bat")   // ссылка на объект файла
var size = fl.Size                          // объем файла
C:\autoexec.bat
WScript.Echo(size)                          /* вывод окна с сообщением
                                               об объеме файла */
```

Значение свойства Size объекта файла нужно сравнить со значением свойства FreeSpace объекта диска, чтобы выяснить, достаточно ли места для записи файла.

5.4.3. Чтение данных из файла и запись данных в файл

Текстовые файлы создаются, чтобы хранить в них данные, и существуют, чтобы содержащиеся в них данные можно было прочесть. Чтение и запись данных производится с помощью следующих трех этапов:

- открытие файла;
- чтение или запись данных;
- закрытие файла.

Открытие текстового файла производится с помощью метода **OpenTextFile** объекта FileSystemObject либо с помощью метода **OpenAsTextStream** объекта файла. При этом файл может быть открыт в трех режимах: только для чтения (for reading only), для записи (for writing) и для добавления (for appending) данных. Режимы для записи и добавления данных не допускают чтения. В режиме добавления записываемые данные добавляются к уже существующим. В режиме записи старые данные теряются, а новые записываются, поэтому режим записи лучше называть режимом перезаписи.

Открытие файла можно осуществить следующими способами:

```
var fso = new ActiveXObject("Scripting.FileSystemObject")
var myfile = fso.OpenTextFile(filepath, mode)
```

либо

```
var fso = new ActiveXObject("Scripting.FileSystemObject")
var fileobj = fso.GetFile(filepath)
var myfile = fileobj.OpenAsTextStream(mode)
```

Здесь filepath — имя файла, возможно, с указанием пути к нему (например, **"C:\\Мои документы\\testfile.txt"**); mode — режим открытия файла:

- 1 — только для чтения (forread only);
- 2 — для записи (for writing);
- 8 — для добавления (for appending).

Заметим, что в результате создания нового текстового файла он остается открытым. Чтобы он был сразу же доступен для записи, необходимо передать методу CreateTextFileQ второй параметр со значением true:

```
var fso = new ActiveXObject("Scripting.FileSystemObject")
fso.CreateTextFile(filepath", true)
```

Для чтения данных из открытого текстового файла используются следующие методы объекта файла:

- Реад(количество_ байтов) — применяется для чтения заданного количества байтов (символов), которое указывается в качестве параметра;
- **ReadLineQ** — применяется для чтения строки, при этом исключается символ перехода на новую строку;
- **ReadAUQ** — применяется для чтения всего содержимого текстового файла.

Если вы используете методы Peacl(количество_байтов) или ReadlineQ и хотите пропустить заданное количество байтов или строку, то можете использовать методы Skip(количество_байтов) и **ScipLineQ** соответственно. Эти методы перемещения по файлу изменяют положение так называемого указателя, которое характеризуется значениями свойств Column (позиция в строке) и Line (номер строки) объекта файла. При первоначальном открытии файла эти свойства, доступные только для чтения, имеют значения 1. Каждое применение методов ReadLineQ и SkipLineQ увеличивает значение свойства Line на 1.

Пример

```
var filepath = "C:\\autoexec.bat"
var fso = new ActiveXObject("Scripting.FileSystemObject")
var myfile = fso.OpenTextFile(filepath, 1)        // объект файла
WScript.Echo(myfile.Line + ", " + myfile.Column ) // 1, 1
myfile.SkipLine()                                 // пропустить строку
WScript.Echo(myfile.Line + ", " + myfile.Column)  // 2, 1
myfile.Skip(14)                                   // пропустить 14 байтов
WScript.Echo(myfile.Line + ", " + myfile.Column ) // 2, 15
```

Заметим, что применение метода ReadAU() после методов перемещения даст в результате содержимое файла, начиная с текущего положения указателя и до конца файла.

Для записи данных в открытый текстовый файл используются следующие методы объекта файла:

- Write (строка) — применяется для записи строки символов без символа перехода на новую строку;
- WriteLine(cTpOKa) — применяется для записи строки символов с добавлением символа перехода на новую строку;
- WriteBlankLines(K(WM4ecTBo) — применяется для добавления пустых строк, количество которых указывается в качестве параметра; по существу, этот метод просто записывает заданное количество символов перехода на новую строку.

Код в листинге 5.4 демонстрирует применение методов записи и чтения данных.

Листинг 5.4. Код методов записи и чтения данных

```
var filepath = "C:\\Мои документы\\1e51Tile.htm"
var fso = new ActiveXObject("Scripting.FileSystemObject")
var myfile = fso.CreateTextFile(filepath, true)  // создание в режиме
записи myf 11e.Wri teLine("<html>")
myfile.WriteLine("<hl>npHBeT всем!</Ы>")
myfile.WriteLine("Это - пробная запись в файл")
myfile.Write("<script>alert('Какое-нибудь сообщение')</script>")
myf ile.WriteBlankLines(2)                  // две пустые строки
myfile.Close()                              // закрываем файл

var fso = new ActiveXObject("Scripting. FileSystemObject")
var myfile = fso.OpenTextFile(filepath,8)   /* открываем в режиме
                                               добавления */
myfile.Write("</html>")                     // дописываем
myfile.Close()                              // закрываем файл

var fso = new ActiveXObject("Scripting.FileSystemObject")
var myfile = fso.OpenTextFile(filepath,1)   // открываем в режиме чтения
myfile.Read(6)                              // читаем б байтов: "<html>"
myfile.SkipLine()                           // пропускаем строку
var x = myf i le. ReadLineO                 /* читаем третью строку:
                                               "<Ы>Привет BceM!</hl> */
myfile.Close()                              // закрываем файл
```

Для проведения экспериментов с файловыми операциями полезно выражения с методами чтения данных передать в качестве параметра методу отображения сообщений WScript.EchoQ. Например, WScript.Echo(myfile.Readl_ine()).

Мы можем прочитать файл построчно в массив, чтобы потом обработать его с помощью методов массива и строкового объекта. В листинге 5.5 приведен код функции, которая читает текстовый файл и возвращает массив.

Листинг 5.5. Код функции, которая читает текстовый файл и возвращает массив

```
function FileToArray(filepath){     // чтение файла в массив
var fso = new ActiveXObject("Scripting. FileSystemObject")
var file = fso.GetFile(filepath)
var size = file.Size                // объем файла
file = fso,OpenTextFile(filepath)
file.Skip(size)                     // перемещаем указатель в конец файла
var lines = file.Line-1             // количество строк
file.Closet)
var fso = new ActiveXObject("Scripting. FileSystemObject")
var file = fso.OpenTextFile(filepath)
var x = new ArrayO
```

продолжение &

Листинг 5.5 *(продолжение)*

```
var i = 0
while(i < linesH{
x[i] = file.ReadMne ()            // чтение строк файла
i++
}
file.CloseO
return x                           // возвращаем массив строк файла
}
```

Чтобы протестировать эту функцию, выполните следующее выражение:

```
WScript .Echo (FileToArray ("C: \\autoexec.bat"))
```

Обратите внимание, что в функции FileToArrayQ нам пришлось дважды создавать объект FSO и открывать файл. Первый раз это понадобилось, чтобы узнать количество строк в файле путем перемещения указателя на величину, равную объему файла. При этом указатель оказывается за пределами файла, и, следовательно, количество строк оказывается на 1 меньше, чем значение свойства Line. К сожалению, мы не можем переместить указатель в начало файла. Поэтому мы закрываем файл и вновь его открываем, чтобы начать построчное чтение и запись в элементы массива с помощью оператора цикла.

5.4.4. Создание ярлыков

Ярлык (значок) представляет собой файл с расширением lnk, содержащий ссылку на некоторое приложение или документ, а также параметры его открытия в окне. При щелчке на ярлыке указанное в нем приложение открывается. Аналогичный ссылочный файл с расширением url содержит URL-адрес документа (веб-страницы). Создать ярлык на рабочем столе компьютера, в меню Пуск, в папке Автозагрузка, Избранное или в любой другой папке можно с помощью объекта Wscript. Shell, входящего в состав WSH. Этот объект (точнее, его экземпляр) создается двумя способами, так же, как и FSO.

- Первый способ:
  ```
  var Myshell = new ActiveXObject("WScript. Shell")
  ```
- Второй способ:
  ```
  var Myshell = WScript .CreateObject ("WScript. Shell")
  ```

С помощью метода SpecialFoldersQ можно узнать местоположение специальных папок, таких как Рабочий стол, Автозагрузка, Избранное, Мои документы, Программы и др. За этими папками закреплены специальные идентификаторы (переменные). Например, папке Рабочий стол соответствует идентификатор Desktop, папке Мои документы — MyDocuments, папке Избранное — Favorites. Вот список всех идентификаторов специальных папок: AUUsersDesktop, AUUsersStartMenu, AUUsersPrograms, AllUsersStartup, Desktop, Favorites, Fonts, MyDocuments, NetHood, PrintHood, Programs, Recent, SendTo, StartMenu, Startup, Templates.

Чтобы получить местоположение папки, например Главное меню, открываемой при щелчке на кнопке Пуск, следует выполнить следующие выражения:

```
var Myshell = new ActiveXObject("WScript. Shell")
var mypath = Myshell.SpecialFolders("StartMenu")
                        /* Значение: C:\Windows\rnaBHoe меню */
```

В листинге 5.6 приводится пример программы создания ярлыка для Блокнота Windows (notepad.exe) и расположения его на рабочем столе компьютера (в папке Рабочий стол).

Листинг 5.6. Код программы создания ярлыка для Блокнота

```
var Myshell = new ActiveXObject("WScript.Shell")
var mypath =
  Myshell.SpecialFolders("Desktop")          // путь к папке Рабочий стол
/* Создание ярлыка и подписи к нему: */
var myshortcut = Myshell.CreateShortcut(mypath + "\\Мой Блокнот.Ink")
/* Папка расположения Windows: */
var mywindir = Myshell.ExpandEnvironmentStrings("%windir%")
/* Параметры ярлыка: */
// расположение файла:
myshortcut.TargetPath = Myshell.ExpandEnvironmentStrings(mywindir +
  "\\notepad.exe") myshortcut .Worki ngDi rectory =
  Myshell.ExpandEnvironmentStrings(mypath)     // рабочая папка
myshortcut.WindowStyle =4                      // тип окна (стандартное)
/* файл, содержащий графическое изображение ярлыка: */
myshortcut.IconLocation = Myshell.ExpandEnvironmentStrings(mywindir +
  "\\notepad. exe")
myshortcut.Save()                              // сохранить на диске
```

Здесь Myshell.ExpandEnvironmentStrings("%windir%") возвращает строку, содержащую значение переменной среды, в данном случае %windir%. По умолчанию файл notepad.exe находится в папке расположения операционной системы Windows, но эта папка не обязательно называется C:\Windows. Свойство WindowStyle может принимать три значения: 3 — развернуть окно на весь экран, 4 — стандартное окно, 7 — свернуть в значок на панели задач.

8 следующем примере создается ярлык в главном меню кнопки Пуск для некоторой программы afpwin.exe:

```
var Myshell = new Acti veXObject ("WScri pt. Shell")
var mypath=Myshell.SpecialFolders("StartMenu")
var myshortcut = Myshell. CreateShortcut (mypath + "\\АФП . Ink")
myshortcut.TargetPath =
  Myshell. ExpandEnvi ronmentStri ngs ("C: \\AFP\\afpwin.exe")
myshortcut.WorkingDirectory =
  Myshell.ExpandEnvi ronmentStri ngs("C: \\AFP")
myshortcut.WindowStyle = 3          // тип окна (развернуть на весь экран)
myshortcut.IconLocation =
  Myshell. ExpandEnvi ronmentStri ngs ("C : \\AFP\\afp.ico")
myshortcut. Save()
```

Приведенный ниже код создает в папке Избранное ссылку (url-файл) на главную страницу веб-сайта автора этой книги:

```
var Myshell = new Acti veXObject ("WScript. Shell")
var mypath = Myshell.SpecialFolders("Favorites")
var myshortcut = Myshell .CreateShortcut (mypath +
  "\\Сам себе Web-дизайнер.url")
myshortcut.TargetPath = Myshell.ExpandEnvironmentStringsC 'http://
  www.admiral.ru/~dunaev")
myshortcut .SaveO
```

Основное отличие этого примера от предыдущих состоит в том, что свойству TargetPath (путь к цели) присваивается URL-адрес документа.

5.4.5. Запуск приложений

Для запуска приложений служит метод **Run()** объекта **Wscript.Shell.** Командная строка запуска приложения (обычно это просто полное имя файла программы) передается методу в качестве строкового параметра.

Примеры

```
var Myshell = new ActiveXObject("WScript.Shell")
My shell. Run ("winword .exe C: \\My\\my document .dope")
Myshell .Run("C: \\MyFolder\\myprogram.exe")
```

5.5. Работа с реестром

Реестр Windows представляет собой базу данных, содержащую сведения об ее настраиваемых параметрах, или, как еще говорят, о конфигурации операционной системы. Кроме того, в реестре хранится информация о настройках аппаратных средств компьютера и программ. Реестры различных систем и версий семейства Windows частично различаются. Однако между ними много общего. В Windows 3.1 (Windows for WorkGroups) реестр хранится в файле reg.dat в папке Windows. В более поздних версиях, Windows 9x/Me и т. д., реестр размещается в двух файлах, расположенных в папке Windows: system.dat и user.dat. Реестр задумывался для замены настроечных **ini-файлов.** Записи в файле реестра имеют более удобную древовидную структуру. Хотя **ini-файлы** также поддерживаются Windows, разработчикам программного обеспечения рекомендуется хранить настроечную информацию в реестре.

Чтобы работать с реестром, необходимо понимать его структуру. Древовидная структура реестра представляет собой иерархически упорядоченное множество разделов (папок), содержащее следующие шесть разделов самого верхнего (корневого) уровня (табл. 5.1).

Таблица 5.1. Разделы реестра

Имяразделареестра верхнего уровня	Сокращенное имя раздела реестра	Описание
HKEY_CLASSES_ROOT	HKCR	Содержит информацию о зарегистрированных типах файлов, OLE и др.
HKEY_CURRENT_USER	HKCU	Содержит параметры настройки оболочки Windows для пользователя, вошедшего в Windows. Например, настройки Рабочего стола, меню кнопки Пуск. Если на компьютере работает единственный пользователь и используется обычный вход в Windows, то содержимое этого раздела берется из подраздела HKEY_USERS\DEFAULT
HKEY_LOCAL_MACHINE	HKLM	Содержит информацию об установленных драйверах и программном обеспечении

Имя раздела реестра верхнего уровня	Сокращенное имя раздела реестра	Описание
HKEY_USERS	HKU	Содержит параметры настройки оболочки Windows для всех пользователей. Информация из этого раздела копируется в раздел HKEY_CURRENT_USER. С другой стороны, все изменения в разделе HKEY_CURRENT_USER автоматически переносятся в раздел HKEY_USER
HKEY_CURRENT_CONFIG	HKCC	Содержит информацию о настройках устройств Plug&Play, а также сведения о настройках компьютера с переменным составом аппаратных средств
HHEY DYN_DATA	HKDD	Содержит изменяющиеся данные о состоянии устройств, установленных на компьютере пользователя. Эти сведения отображаются в окне Панель управления > Система > Устройства > Свойства. Данные этого раздела обновляются операционной системой Windows, и поэтому не рекомендуется редактировать его самостоятельно

Перечисленные коренные разделы имеют подразделы, а те — свои подразделы и т. д. В конечном разделе ветви дерева реестра определяются параметры. Каждый параметр имеет имя и значение. Работа с реестром заключается в просмотре, создании и удалении его записей. Так, вы можете создать или удалить раздел реестра, создать или удалить параметр в каком-либо разделе. Однако делать это без четкого понимания целей и правил не рекомендуется.

ВНИМАНИЕ

Не следует открывать и редактировать файлы реестра system.dat и user.dat в текстовом редакторе.

Здесь мы не будем подробно обсуждать, что нужно изменить в реестре, чтобы получить тот или иной эффект. Рассмотрим, как можно изменять записи в реестре, если такая необходимость возникла.

Для работы с реестром используются три основных способа.

- С помощью редактора реестра — программыregedit.exe. Чтобы запустить редактор реестра, достаточно выполнить команду Пуск > Выполнить, ввести с клавиатуры слово regedit и щелкнуть на кнопке ОК. Данный способ наиболее безопасный.
- С помощью reg-файлов. Это текстовые файлы с расширением гед, записи в которых имеют довольно простую структуру. Запуск reg-файла, например, двойным щелчком на его имени в Проводнике Windows приводит к открытию диалогового окна с предложением добавить информацию из этого файла в реестр. При вашем согласии данные из reg-файла будут импортированы в файлы реестра.

- С помощью программы JavaScript, использующей специальные методы Windows Scripting Host. В этом случае можно организовать не видимую пользователем работу с реестром (если, конечно, он не заблокировал выполнение сценариев).

Рассмотрим сначала структуру записей в reg-файле, который можно создать с помощью обычного текстового редактора, например Блокнота Windows. В первой строке этого файла должно быть написано прописными буквами REGEDIT4. Затем должна быть пустая строка. В следующей, третьей строке в квадратных скобках пишется имя раздела реестра. В четвертой строке располагается запись согласно приведенному ниже формату:

```
"имя_параметра"=значение
```

Если в данном разделе реестра следует разместить еще один параметр, то запись о нем располагается в следующей строке. Таким образом, сведения о каждом параметре записываются в отдельной строке reg-файла. Аналогичным образом можно создать записи, относящиеся к другому разделу реестра. Однако между такими секциями, каждая из которых соответствует отдельному разделу реестра, обязательно должна быть одна пустая строка. Таким образом, структура reg-файла имеет следующий вид:

```
REGEDIT4

[имя_раздела1]
"имя_параметраНазначение11
"имя_параметра12"=значение!2
. . .
"имя_параметра1M"=значение1M

[имя_раздела2]
"имя_параметра21"=значение21
"имя_параметра22"=значение22
. . .
"имя_параметра2M"=значение2M
. . .
[имя_раздела1_]
"имя_лараметра1Л"=значение1_1
"имя_параметра1_2"=значение1-2
. . .
"имя_параметра1_M"=значение1_M
```

Значения параметров могут принадлежать одному из трех типов:

- строковый — значения этого типа являются просто строкой символов, заключенной в кавычки;
- DWORD — для записи значения этого типа используется формат dword:XXXXXXXX. Вместо X записываются шестнадцатеричные цифры. Обычно параметры типа DWORD имеют значение либо 0, либо 1. В этих случаях для задания значения требуется записать либо dword:00000000, либо dword:00000001;
- двоичный — для записи значения этого типа используется формат hex:XX,XX,XX,... Вместо XX записываются шестнадцатеричные цифры; пары таких цифр разделяются запятой. Например, для задания значения af 00 01 00 следует записать hex:af,00,01,00.

Кроме того, в реестре могут быть установлены параметры по умолчанию (default). Чтобы присвоить какое-то значение параметру по умолчанию, необходимо просто записать в reg-файле следующее выражение:

```
(Э=значение
```

Рассмотренные выше записи reg-файла добавляют, а не перезаписывают записи в реестре. Чтобы удалить раздел в реестре, необходимо в reg-файле перед его именем в квадратных скобках поставить символ «минус», как в следующем примере:

```
[-HKEY_LOCAL_MACHINE\Software]
```

Чтобы удалить параметр, следует присвоить ему символ «минус»:

```
"имя_параметра"= -
```

Ниже в качестве примера приведено содержимое reg-файла, с помощью которого устанавливается начальная веб-страница, загружаемая в браузер Internet Explorer:

```
REGEDIT4

[HKEY_CURENT_USER\SOFTWARE\MicrosoftXInternet Explorer\Main]
"StartPage" = "http://www.myweb.ru"
```

Читать, создавать и удалять записи в реестре можно с помощью специальных методов объекта **Wscript.Shell,** входящего в состав WSH:

- RegReadQ — возвращает запись реестра или значение параметра;
- RegWriteQ — создает новую запись в реестре или изменяет значение параметра уже существующей записи;
- **RegDeleteQ** — удаляет запись реестра или параметр.

Применение этих методов имеет следующий синтаксис:

```
var Myshell = new ActiveXObject("WScript.Shell")
Myshell.метод(параметры)
```

Метод RegWrite () принимает в качестве параметра строку, содержащую имя раздела реестра, за которым указывается имя параметра. Все элементы имени раздела разделяются двойными обратными слэшами. Например, в строке **"HKEY_CURENT_USER\\Myreg\\myparam"** последний элемент myparam является именем параметра, а не раздела реестра.

Метод RegWriteQ принимает три параметра, из которых последний не является обязательным. Первый параметр — строка, содержащая имя раздела или имя параметра. Если эта строка заканчивается двойным слэшем, то подразумевается, что последний элемент строки — имя раздела, в противном случае — имя параметра в разделе. Второй параметр метода RegWriteQ представляет значение параметра раздела. Если имя параметра раздела не указано, то подразумевается параметр по умолчанию. Третий, необязательный параметр метода RegWriteQ задает тип параметра в разделе реестра и представляет собой **"REG_DWORD"** или **"REG_BINARY",** соответственно для типов DWORD и двоичного. Если тип не указан, то подразумевается строковый тип.

Примеры

```
var Myshell = new ActiveXObject("WScript.Shell")
/* Создание подраздела Myreg в разделе HKEY_CURENT_USER и присвоение
   параметру по умолчанию значения "значение": */
Myshell.RegWrite("HKEY_CURENT_USER\\Myreg\\", "значение")
```

```
/* Создание в разделе HKEY_CURENT_USER\ Myreg строкового параметра
   myparaml и присвоение ему значения "некоторая строка": */
Myshell.RegWrite("HKEY_CURENT_USER\\Myreg\\myparaml", "некоторая
  строка")
/* Создание в разделе HKEY_CURENT_USER\ Myreg двоичного параметра
   myparam2 и присвоение ему значения 5: */
Myshell.RegWrite("HKEY_CURENT_USER\\Myreg\\myparam2", 5, "REG_BINARY")
/* Создание в разделе HKEY_CURENT_USER\ Myreg параметра typaram3 типа
   DWORD и присвоение ему значения 3: */
Myshell.RegWrite("HKEY_CURENT_I)SER\\Myreg\\myparam3", 3, "REG_DWORD")
```

Метод **RegDeleteQ** принимает в качестве параметра строку, содержащую имя раздела или параметра. Если эта строка заканчивается двойным слэшем, то подразумевается, что последний элемент строки — имя раздела, в противном случае - имя параметра в разделе.

Примеры

```
var Myshell = new ActiveXObject ("WScript. Shell")
/* Удаление в разделе HKEY_CURENT_USER\ Myreg параметра myparaml: */
Myshell.RegWrite("HKEY_CURENT_USER\\Myreg\\myparami")
/* Удаление подраздела Myreg в разделе HKEY_CURENT_USER: */
Myshell.RegDelete("HKEY_CURENTJJSER\\Myreg\\")
```

СОВЕТ

Вместо полных имен корневых разделов реестра можно использовать их сокращенные обозначения. Например, вместо **HKEY_CURENT_USER** можно писать HKCU.

В листинге 5.7 приведен пример программы, делающей запись в реестре, которая устанавливает стартовую веб-страницу.

Листинг 5.7. Запись установки стартовой страницы

```
var Myshell = new ActiveXObject("WScript.Shell")
var mystartpage = http://www.myweb.ru    // нужная страница
/* Проверяем, какая веб-страница является начальной: */
startpage = Myshell. RegWri te("HKCUX\SOFTWARE\\Mi crosoft\\Internet
  ExplorerUMainUStar' Page")
if (startpage != mystartpage)           // если другая
   Myshell. RegWri te( "HKCU\\SOFTWARE\\Mi crosoft\\Internet
Explorer\\Main\\Start Pagi" . mystartpage)
```

Аналогичным образом можно создать записи в реестре, запускающие приложения.

Приложение 1. Руководство по динамическому HTML

Основные понятия

HTML (HyperText Markup Language — язык разметки гипертекста) является основным языком программирования веб-страниц. Описания веб-страниц содержатся в HTML-программах (HTML-кодах), которые хранятся в обычных текстовых файлахс расширением htm или html. Иногда эти программы называют HTML-документами, но обычно HTML-документом считается то, что можно видеть в окне браузера. Программы на языке HTML содержат инструкции (коды), называемые тегами. Теги представляют собой последовательности символов, заключенные в угловые скобки (например, <P>).

Большинство современных браузеров допускают запись тегов в любом регистре, то есть как прописными, так и строчными буквами. Все ключевые слова тегов, являющиеся обычно аббревиатурами слов английского языка, записываются буквами латинского алфавита. Например, IMG —сокращение слова image (изображение).

HTML-программа должна начинаться тегом <HTML> и заканчиваться тегом </HTMl_>. Между ними находятся другие теги программы или текст, который вы хотите вывести в окне браузера. Некоторые теги используются только парами (например, <HTMl_> и </HTML>). При этом первый из них называется открывающим, а второй — закрывающим. Иногда парные теги называют контейнерными, потому что между ними можно разместить другие теги. Таким образом, в контейнерные теги можно вкладывать другие теги, в том числе и контейнерные, то есть теги могут быть вложенными. Сейчас запомните этот факт, а в дальнейшем разберемся, как ими пользоваться. Существуют одиночные теги, для них нет соответствующих закрывающих тегов. Примером одиночного тега является тег
 (конец строки).

Теги могут содержать параметры, называемые атрибутами, которые, в свою очередь, могут иметь значения — аргументы. Можете считать (если так удобнее), что тег — это команда, атрибут — имя ее параметра, а аргумент — значение параметра. Например, для вывода на экран изображения, хранящегося в файле картинка.]рд, используется тег

```
<IMG SRC="КарТННКа.j pg">
```

Здесь IMG — название тега, SRC — атрибут, а "картинка.рд" — аргумент атрибута SRC.

Итак, если мы решили написать HTML-программу, то должны включить в нее два тега:

```
<HTML>
    ...          (здесь будут другие теги программы)
</HTML>
```

HTML-программы состоят из двух основных частей: заголовка и тела. Каждая из этих частей ограничивается соответствующей парой тегов. Так, заголовок ограничивается парой тегов <HEAD>И </HEAD>, а тело — тегами <BODY> и </BODY>. В результате HTML-программа выглядит следующим образом:

```
<HTHL>
   <HEAD>
      ... (здесь будет заголовок)
   </HEAD>
   <BODY>
      ... (здесь будут теги тепа программы)
   </BODY>
</HTML>
```

Заметим, что писать каждый тег с новой строки или делать отступы совсем не обязательно, однако благодаря этому программа читается лучше.

Между тегами <HEAD> и </HEAD>, обрамляющими заголовок программы (HTML-файла), напишем еще два тега: <TITLE> и </TITLE>. С помощью этих тегов обрамляется текст, который помещается в заголовок браузера, то есть в самую верхнюю полоску его окна. Пусть текст заголовка будет, например, таким: «Основные элементы HTML». Тогда программа примет следующий вид:

```
<HTML>
   <HEAD> <TITLE> Основные элементы HTML </TITLE> </HEAD>
   <BODY>
      ... (здесь будут расположены теги тела программы)
   </BODY>
</HTML>
```

Заметьте, что тег <TITLE> вложен в тег <HEAD>, а тот вложен в тег <HTML>.

В теле нашей программы, то есть между тегами <BODY> и </BODY>, напишем какой-нибудь текст, чтобы программа выглядела следующим образом:

```
<HTML>
   <HEAD> <TITLE> Основные элементы HTML </TITLE> </HEAD>
   <BODY>
HTML-документы состоят из заголовка и тела. В теле документа могут
находиться тексты, рисунки и ссылки на другие файлы.
   </BODY>
   </HTML>
```

Раскроем окно текстового редактора Блокнот (Notepad) и напишем в нем текст нашей первой HTML-программы (рис. П1.1).

Сохраним нашу программу в файле с расширением htm или html. Например, назначим имя файла проба.htm. Чтобы удалось назначить нужное расширение файла, выберите в диалоговом окне тип файла Все файлы (*.*) и введите в поле Имя файла придуманное вами имя файла с нужным расширением. Теперь необходимо

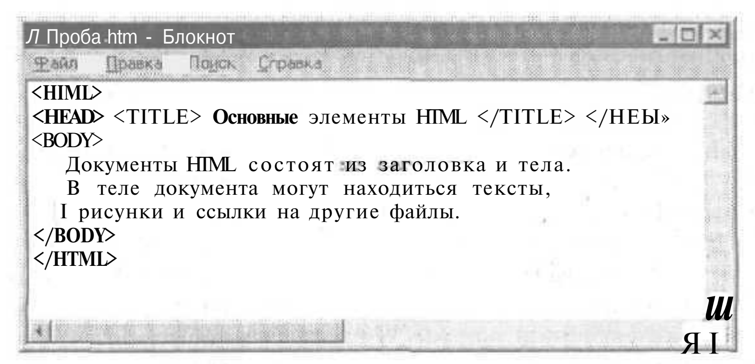


Рис. П1.1. HTML-код в окне текстового редактора Блокнот (Notepad)

раскрыть этот файл в браузере. Для этого в Проводнике Windows (Explorer) найдите файл проба.htm и выполните двойной щелчок левой кнопкой мыши на его имени или на его значке (рис. П1.2). В Windows все файлы с расширением htm и html открываются в окне браузера Internet Explorer, если, конечно, вы не установили другой браузер.

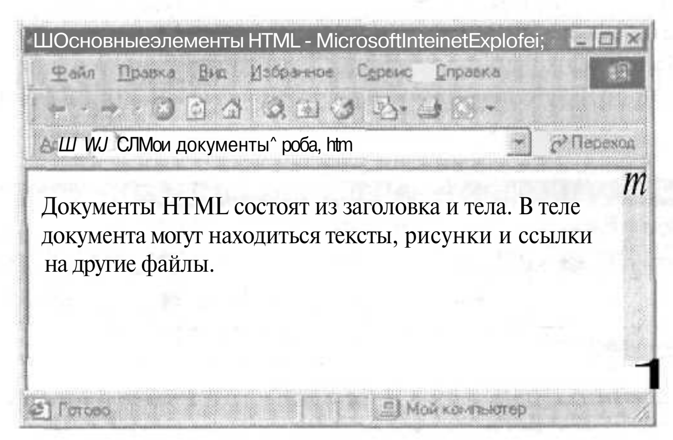


Рис. П1.2. HTML-документ в окне браузера Internet Explorer

Если потребуется что-то изменить в HTML-коде, то, не выходя из окна браузера, в меню Вид выберите команду (опцию) Просмотр HTML-кода. В результате раскроется окно текстового редактора Блокнот с файлом вашей программы. Внесите необходимые изменения и сохраните файл под тем же именем. После этого в окне браузера выполните команду Вид > Обновить. В результате загрузится измененный HTML-файл.

Итак, мы создали простую страницу. В ней размещен только текст, причем неформатированный. Однако настоящие страницы устроены сложнее и выглядят более привлекательно. Более того, в настоящее время в Интернете существуют удивительные по красоте страницы. И это очень важно, поскольку необходимо не просто опубликовать свою страницу, содержащую информацию, но еще и привлечь к ней внимание читателей.

Многие пользователи привыкли работать с солидными текстовыми редакторами (текстовыми процессорами), например MS Word. Конечно, для написания текстов HTML-программ можно использовать любимые редакторы. Однако при этом могут возникнуть проблемы. Например, MS Word распознает (не только по расширению файла, но и по содержанию), что файл содержит HTML-программу, и показывает не ее текст (код), а результат работы программы, то есть то, как будет выглядеть этот HTML-документ в окне браузера. Поэтому если вы пользуетесь MS Word, сохраняйте текст HTML-программы как текстовый файл в формате txt, а затем переименовывайте его в файл с расширением htm или html. Всего этого можно избежать, если сразу использовать простой текстовый редактор Блокнот (Notepad). Используя Блокнот, а не MS Word, вы будете уверены, что все несуразности отображения HTML-документа в браузере определяются вашей HTML-программой, а не «интеллектом» текстового процессора. Когда вы изучите основы языка HTML, MS Word, а также другие средства автоматизации создания HTML-документов (например, FrontPage) действительно облегчат программирование.

Вместо простого текстового редактора вы можете использовать специальные редакторы веб-страниц, например Microsoft FrontPage или Macromedia Dreamweaver. При выборе средств решается следующая дилемма. Любителям, занимающимся веб-дизайном не систематически, прежде всего нужен конечный результат. Профессионалы же любят все держать под контролем, им важен не только эффект, но и способ, которым он получен. Хотя данная книга написана, в первую очередь, для любителей, все же постараемся следовать принципам профессионалов.

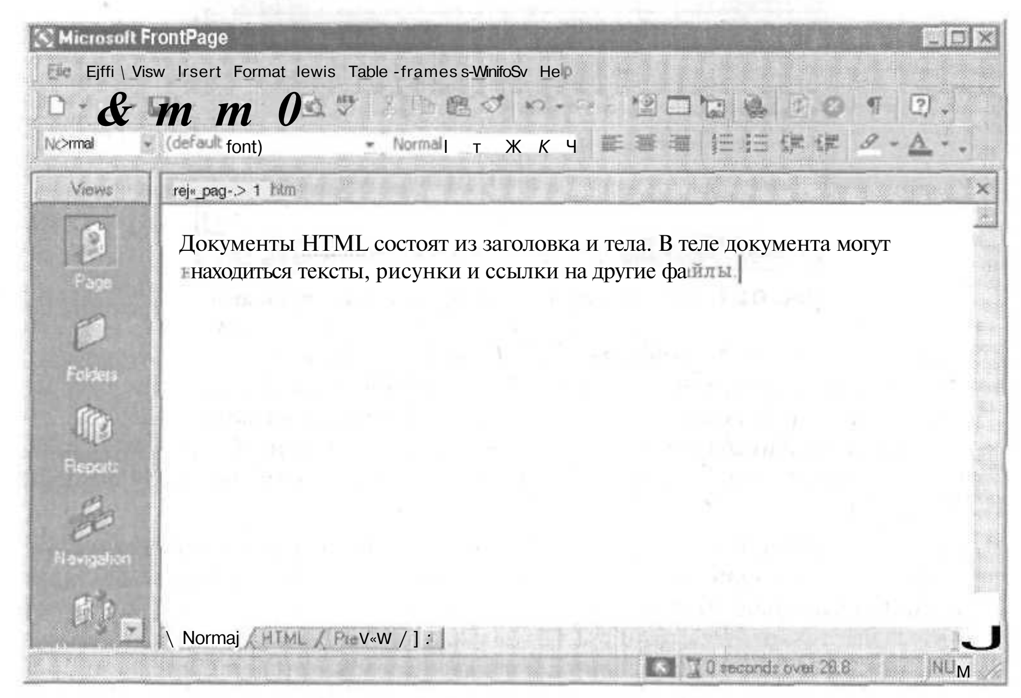


Рис. П1.3. Окно программы FrontPage

В случае использования такого средства автоматизации разработки веб-страниц, как FrontPage, всю описанную выше работу можно выполнить следующим образом. Запустите FrontPage. Выберите вкладку Normal. В рабочую область, расположенную в центре окна, введите с клавиатуры текст. У вас должно получиться примерно то, что показано на рис. П1.3.

Перейдите на вкладку HTML, чтобы увидеть, какой код программы соответствует этой простой странице, содержащей только текст (рис. П1.4). Этот HTML-код можно при желании отредактировать. Чтобы посмотреть результат редактирования, достаточно просто перейти на вкладку Normal.

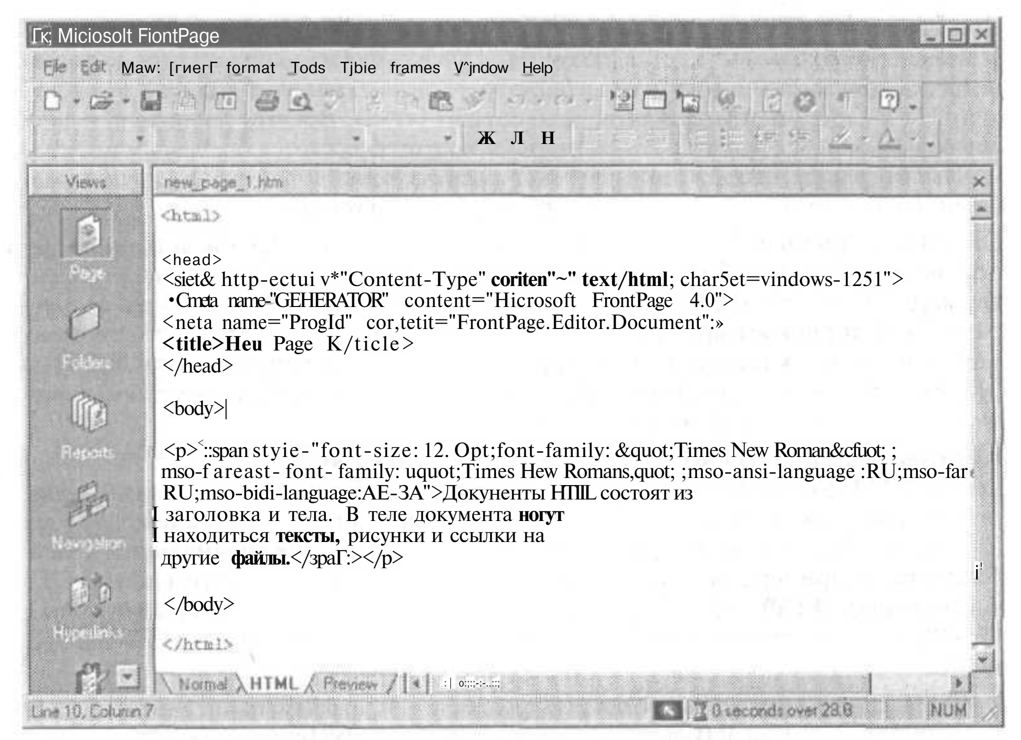


Рис. П1.4. Код на вкладке HTML в программе FrontPage

Если вы хотите посмотреть, как будут выглядеть результаты вашего творчества в окне браузера, перейдите на вкладку Preview.

Как FrontPage, так и MS Word при создании HTML-программ вставляют в заголовок (то есть между <HEAD> и </HEAD>) теги <META> с различными атрибутами.

Пример

```
<HTM1_>
<HEAD>
<META http-equiv="Content-Language" content="ru">
<META http-equiv="Content-Type" content="text/html" ; charset="windows-
1251">
<META name="GENERATOR" content="Microsoft FrontPage 4.0">
<METAname="ProgID" con t en t = " Front Page . Editor . Document">
```

```
</HEAD>
<BODY>
 ...
</BODY>
</HTML>
```

Информация, содержащаяся в тегах <META>, не отображается браузером и служит специальным целям. Например, в ней указываются язык, на котором написан документ, кодовая страница, ключевые слова для поиска и др. Эти данные очень важны для настройки браузера и работы поисковых серверов. Теги <META> можно вставить в HTML-программу на заключительном этапе разработки веб-страницы, называемом публикацией.

При написании HTML-программ, особенно если в них содержится много тегов, возникает необходимость вставки комментариев — поясняющих текстов, которые не видны при загрузке документа в браузер. Комментарии необходимы разработчику HTML-документа. Для этой цели служит тег <!>. Все, что заключено между символами <! и >, считается комментарием и не отображается браузером на экране. Имейте в виду, что даже программист-профессионал через месяц с трудом разбирается в своем коде, и тогда комментарии помогают ему.

Обратите внимание на то, что перенос слов текста в окне браузера происходит в зависимости от ширины окна. Если вы хотите, чтобы текст располагался по центру, применяйте тег <CENTER>. Если пользователь изменит размеры окна браузера, произойдет новое выравнивание текста. Заметим, что тег <CENTER> можно применять не только к тексту, но и к графике, таблицам и другим элементам, которые будут рассмотрены ниже. Тег <CENTER> является контейнерным, то есть ему соответствует закрывающий тег </CENTER>.

Некоторым тегам в HTML-программе соответствуют вполне определенные элементы HTM-документа, размещаемые на странице. Например, мы можем написать теги, выводящие на экран такие элементы, как текст, рисунок, видеоклип и др. Другие же теги ничего не выводят. Они предназначены для специальных целей. Типичными примерами таких тегов являются контейнерные теги (или просто контейнеры): <HEAD>, <BODY> и <CENTER>. Отметим здесь еще один контейнерный тег <DIV>, с помощью которого можно выделить часть (или раздел) HTML-документа.

В HTML-программу можно вставлять фрагменты программ, написанных на языке Java Script или Visual Basic Script (только для браузера Internet Explorer). Такие фрагменты называются сценариями и обрамляются тегами <SCRIPT> и </SCRIPT>. В сценариях обычно описывается обработка различных событий, например щелчок клавишей мыши на кнопке или изображении. Использование сценариев описано в основной части этой книги.

Итак, прежде чем двигаться дальше, вы должны четко уяснить следующее.

- Программа на языке HTML хранится в текстовом файле с расширением htm или html.
- HTML-программа состоит из тегов. Теги могут вкладываться друг в друга, иметь атрибуты, которые, в свою очередь, могут принимать значения (аргументы). Теги могут быть контейнерными, то есть парными, и одиночными.
- Некоторым тегам в HTML-программе соответствуют видимые элементы HTML-документа (заголовки, фрагменты текста, рисунки и т. п.), выводимые на эк-

ран в окне браузера. Некоторые теги выполняют специальные функции, не связанные с выводом на экран.

- Все, что заключено между символами <! и >, является комментарием и не отображается в окне браузера.
- Информация, описанная в тегах <META>, не отображается браузером, но имеет важное значение. Эти теги можно вставить в HTML-программу на завершающем этапе разработки веб-страницы, непосредственно перед ее опубликованием.
- В HTML-программу можно вставлять фрагменты, написанные на языках JavaScript и Visual Basic Script. Эти фрагменты обрамляются тегами <SCRIPT> и </SCRIPT>.

Форматирование текстов

Страница обычно содержит тексты — простой и наиболее распространенный способ представления информации, хотя далеко не единственный. Вы можете создать текст, не уделяя особого внимания тому, как он будет выглядеть в окне браузера. Браузер отобразит неформатированный текст, используя настройки, выбранные пользователем. Однако даже самому непритязательному пользователю хочется, чтобы заголовки отличались от основного текста размером и шрифтом, чтобы можно было выделить абзацы, пропустить строку и т. п. Все это называется форматированием текста. Сразу предупредим, что если вы переусердствуете в этом, то пользователь просто не сможет прочитать ваш текст, не говоря уже об эстетическом восприятии. Запомните, что тексты — это главное на вашей странице. Они должны правильно отображаться почти во всех браузерах.

Если вы работаете в специальной программе, предназначенной для создания веб-страниц (например, FrontPage), то у вас имеется множество вариантов, подобных форматированию в обычных текстовых процессорах. Поэкспериментируйте, наблюдая при этом, какие теги создает программа FrontPage (см. вкладку HTML).

При оформлении текстов используются специальные теги. Рассмотрим некоторые из них.

- Тег
 предписывает переход на новую строку.
- Тег <P> является тегом абзаца. После него текст будет выводиться с новой строки и, кроме того, одна строка будет пропущена. Если не использовать эти теги, то разбиение текста на строки будет определяться шириной окна браузера, так что вид текста может оказаться совсем не таким, каким мы его себе представляли.
- Если вы хотите, чтобы текст выравнивался по центру окна браузера, используйте тег<CENTER>, который упоминался выше.

ВНИМАНИЕ

Теги
 и <P> действуют по отношению не только к текстам, но и к другим элементам страницы. Например, если вы хотите, чтобы рисунок размещался ниже текста, то поставьте между текстом и тегом рисунка
 или <P>.

Стандартные логические стили

Для выбора размера шрифта можно использовать теги так называемых логических стилей. Их всего шесть, обычно они используются для определения заголовков различного уровня. При переходе от первого стиля к шестому постепенно уменьшается размер и толщина букв шрифта. Теги логических стилей записываются как <H1>, <H2>, ... <Нб>. Каждый из них имеет соответствующий закрывающий тег. Например, тегу <H1> соответствует закрывающий **тег</H1>.**

Пример

```
<H1>Заголовок 1-го уровня </H1>
```

задает вывод текста Заголовок 1-го уровня шрифтом, соответствующим первому логическому стилю.

Заметим, что логический стиль определяет стиль текста сообразно настройкам браузера. При этом стиль <H2> всегда будет «меньше», чем <H1>, если, конечно, автор страницы не переопределил его по своему усмотрению. Дело в том, что вы имеете возможность переопределить установки по умолчанию. Для этого используются средства каскадных таблиц стилей (CSS).

Следующая программа демонстрирует использование тегов логических стилей (рис. П1.5).

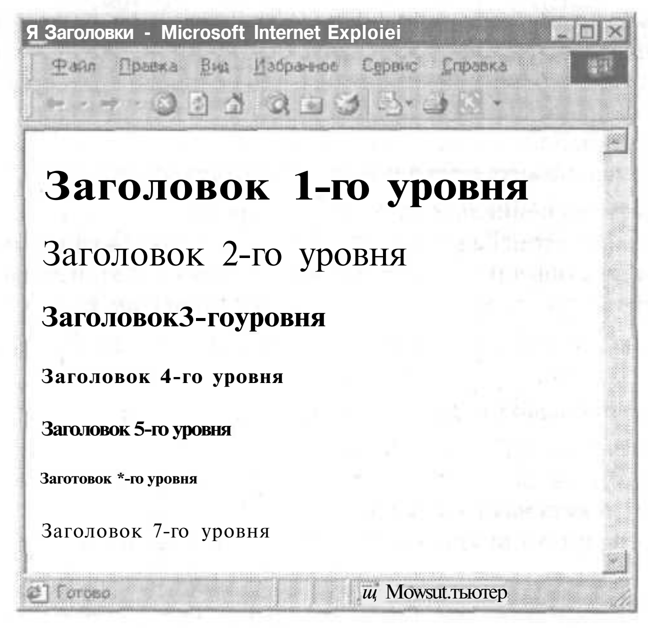


Рис. П1.5. Логические стили

```
<HTHL>
<HEAD> <TITLE> Основные элементы HTML </TITLE></HEAD>
<BODY>
<H1>Заголовок 1-го уровня </H1>
<H2>Заголовок 2-го уровня </H2>
```

```
<H3>Заголовок 3-го уровня </H3>
<H4>Заголовок 4-го \фовня </H4>
• <H5>Заголовок 5-го уровня </H5>
<H6>Заголовок б-го уровня </H6>
<H7>Заголовок 7-го> hровня </H7>
</BODY>
</HTML>
```

Управление шрифтом

Кроме использования стандартных размеров и начертаний (гарнитуры) шрифтов можно определять шрифты для каждого фрагмента текста с помощью специальных тегов.

Самый простой способ — использование так называемых физических стилей (табл. П 1.1).

Таблица П1.1. Теги физических стилей

Стиль	Тег
Полужирный (Bold)	<B>
Курсив (Italic)	<I>
Подчеркнутый (Underscore)	<U>
Вычеркнутый (Strike Out)	<S>
Пишущая машинка (Typewriter)	<TT>
Мерцающий (только для браузера Netscape Navigator)	<BLINK>

Для каждого тега физического стиля существует соответствующий закрывающий тег, который отменяет стиль, установленный ранее. Например, для тега <B> закрывающим тегом является </B>.

Ниже приведен пример программы и внешний вид различных физических стилей (рис. П1.6):

```
<HTM1_>
<HEAO><T1T1E>Физические ст"ли</T1T1E></HEAO>
<BODY>
<B>Полужирный<P></B>
<1>Курсив<P></1>
<и>Подчеркнутый<P></и>
<5>Вычеркнутый<P></5>
<TT>Пишущая машинка<P></TT>
<B><1>Полужирный курсив<P>
<11>Попужирный курсив подчеркнутый
</BODY>
</HTML>
```

Внутри тега заголовка можно вставить тег физического стиля, чтобы модифицировать весь заголовок или только некоторую его часть (рис. П1.7). Например, чтобы выделить курсивом часть текста, определенного в качестве заголовка, можно использовать следующую конструкцию:

```
<HTML>
<HEAO><T1T1_E>Физические и логические стили</T1T1_E></HEAO>
```

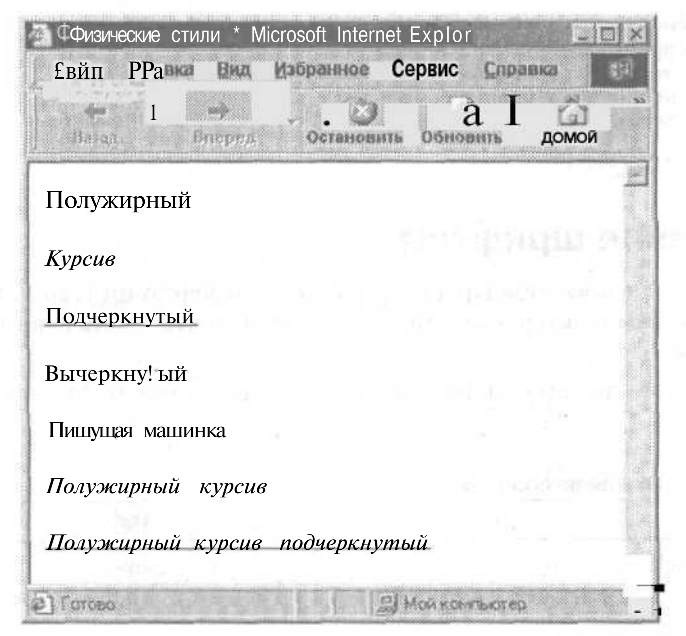


Рис. П1.6. Физические стили

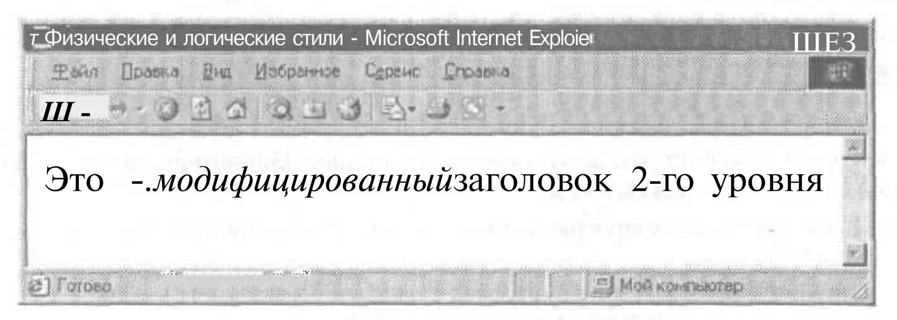


Рис. П1.7. Изменение части заголовка

```
<BODY>
<H2>Это - <I>модифицированный</I> заголовок 2-го уровня</H2>
</BODY>
</HTML>
```

С помощью специального тега <FONT> можно настроить шрифт для изображения текста: задать гарнитуру, размер и цвет. Прежде всего вы можете установить размер основного шрифта, который используется в документе по умолчанию. Тег основного шрифта имеет формат <BASEFONT SIZE=размер_шрифта>.

Размер основного шрифта можно установить от 1 до 7. Если не использовать этот тег, то размер основного шрифта по умолчанию устанавливается равным 3.

Тег <FONT SIZE=размер_шрифта> используется для установки размера текущего шрифта в отдельных фрагментах текста. На стили этот тег не влияет. Диапазон возможных значений — от 1 до 7. Данный тег позволяет также управлять разме-

ром текущего шрифта относительно основного. Для этого используются символ «+» (чтобы увеличить) и символ «-»(чтобы уменьшить размер шрифта на заданную величину). Например, если размер основного шрифта установлен равным 3, то тег <FONT SIZE=+2> устанавливает размер текущего шрифта равным 5.

Для задания гарнитуры шрифта используется тег <FONT PACE="имя_шрифта">.

Пример

```
<FONT FACE="Arial">
```

Если этот тег не используется в вашем HTML-документе, то браузер будет применять шрифт, установленный в настройках. Поэтому текст на экране пользователя может выглядеть совсем не так, как вы себе его представляли. Следует также иметь в виду, что если назначенный вами шрифт не установлен на компьютере пользователя, то браузер будет изображать текст шрифтом, установленным по умолчанию. В теге <FONT> вы можете указать через запятую перечень шрифтов. В этом случае браузер будет использовать первый найденный шрифт.

Пример

```
<FONT FACE="Arial, Sans Serif, Courier">
```

Обычно в перечне задают похожие шрифты. Рекомендуется назначать наиболее популярные шрифты. При размещении на странице текстовой информации лучше вообще не назначать название шрифта, полагаясь на стандартные настройки браузера. Но тогда при разработке страницы следует также использовать стандартные настройки браузера, чтобы синхронизировать свое восприятие текста с возможным восприятием пользователя. В конце концов, вы создаете страницу не для себя, а для читателей.

С помощью атрибута COLOR в теге <FONT> можно задать цвет шрифта:

```
<FONT C01,Ж="цвет">
```

Аргумент атрибута COLOR представляет собой шестнадцатеричную запись кода цвета (красной, зеленой и синей составляющей, или, иначе говоря, RGB-составляющей). Следующая программа демонстрирует управление шрифтом (рис. П1.8).:

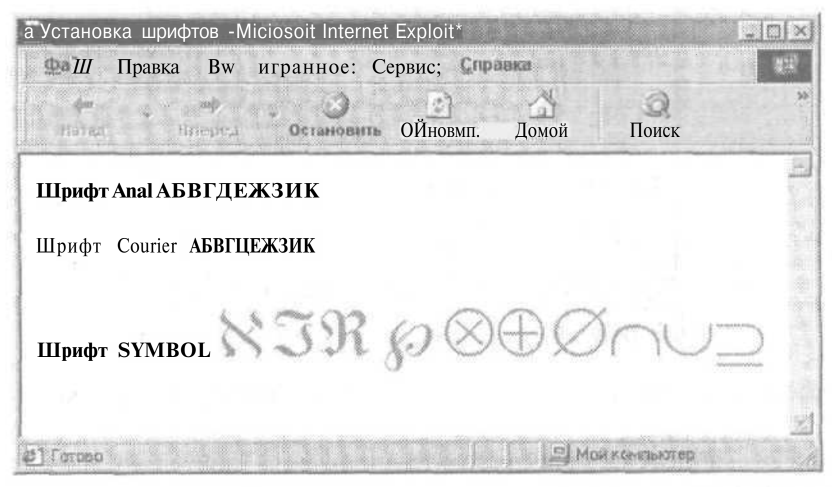


Рис. П1.8. Использование различных шрифтов

```
<HTML>
<HEAO><T1T1_E>Установка шрифтов</TITLE></HEAD>
<BODY>
<P>Шрифт AriaKFONT FACE="ARIAL">АБВГДЕЖЗИК </FONT>
<P>Шрифт Courier<FONT PACE="C01ШЕ1Г>АБВГДЕЖЗИК </FONT>
<P>Шрифт SYMBOL<FONT FACE="SYMBOL" SIZE=7 C010K="ЯЕО">АБВГДЕЖЗИК </FONT>
</BODY>
</HTML>
```

Заметим, что в теге <FONT> можно использовать несколько его атрибутов или все возможные.

Пример

```
<FONT FACE="Arial" SIZE=5 COLOR="BLUE">
```

В математических формулах, а также для подстрочных замечаний часто применяются индексы, которые отличаются от основного текста положением относительно строки (чуть выше или ниже) и размером (рис. П1.9). Для этой цели служат теги <SUP> и <SUB> — соответственно для верхних и нижних индексов.

```
<HTML>
<HEAO><T1T1_E>Индексы< /TITLE></HEAD>
<BODY>
<H3>Пример использования индексов
<P>(5+x<SUP>2</SUP>)<SUP>x+3</5UP>
<P>a<SUB>K/SUB>+ a<SUB>2</SUB>+ a<SUB>3</SUB>
<P>Подстрочные примечания <SUP>2</SUP>
</H3>
</BODY>
</HTML>
```

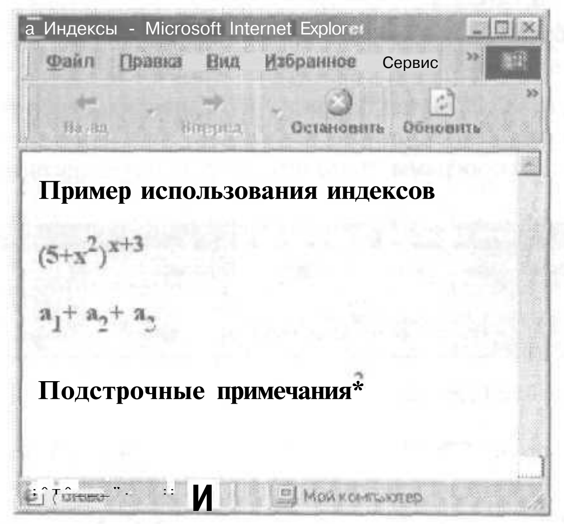


Рис. П1.9. Использование верхних и нижних индексов

Кроме рассмотренных выше, имеются дополнительные теги форматирования текстов:

- <ADDRES> — выделение адресов электронной почты, почтовых адресов и номеров телефонов;

- <CITE> — выделение цитат;
- <CODE>,<SAMP> — запись текстов программ, символьных констант;
- <KBD> — ввод текстов с клавиатуры.

В последних трех стилях используется моноширинный шрифт (обычно Courier). Например, символ «I» данного шрифта занимает столько же места, сколько и буква «Ж». Использование таких шрифтов обусловлено простой возможностью выравнивания текста с помощью символа пробела.

Отметим еще один момент. В теги управления шрифтом, как и в теги логических стилей, можно вставлять атрибут T1T1E="строка", что позволяет привязать к тексту внутри этого тега всплывающую подсказку. Аргументом атрибута TITLE является строка подсказки. При остановке курсора мыши на выделенном слове или фразе около курсора появится подсказка. С помощью этого приема можно расшифровывать аббревиатуры, давать дополнительные пояснения и рекомендации пользователю.

Цвет

По умолчанию браузеры заполняют фон сплошным цветом, определенным настройкой браузера: серым, белым или черным. Пользователи по-разному настраивают цвета, поэтому иногда имеет смысл принудительно зафиксировать цвет фона или создать фоновое изображение. Если вы не позаботитесь об этом, в худшем случае пользователь не сможет прочитать синий текст на черном фоне. Фоновый цвет задается в теге <BODY>.

Цвет фона определяется атрибутом BGCOLOR тега <BODY>. Например, тег, задающий цвет фона "#FF1230", имеет вид:

```
<BODY BGCOLOR="#FF1230">
```

а желтый цвет фона — следующий вид:

```
<BODY BGCOLOR="YELLOW">
```

Можно задать и цвет текста. Для этого служит атрибут TEXT тега <BODY>. Тег, приведенный ниже, задает зеленый цвет фона и синий цвет для текста:

```
<BODY BGCOLOR="GREEN" TEXT="BLUE">
```

Как уже было сказано, цвет можно указывать по имени и шестнадцатеричным кодом. Ниже приводится таблица соответствий некоторых имен цветов и их шестнадцатеричных представлений.

Таблица П1.2. Таблица соответствий имен цветов и их шестнадцатеричных представлений

Цвет	Имя цвета	**Шестиадцатеричное** представление
Черный	BLACK	#000000
	NAVY	#000089
	SILVER	#C0C0C0
Синий	BLUE	#0000FF
	MAROON	#800000
Пурпурный	PURPLE	#800080

продолжение

Таблица П1.2 *(продолжение)*

Цвет	Имя цвета	**Шестнадцатеричное** представление
Красный	RED	#FFOOOO
	FUCHSIA	#FFOOFF
Зеленый	GREEN	#008000
	TEAL	#008080
	LIME	#OOFFOO
	AQUA	#OOFFFF
	OLIVE	#808000
Серый	GREY	#808080
Желтый	YELLOW	#FFFFOO
Белый	WHITE	#FFFFFF

Текст заданного формата

Браузер обычно преобразует текст HTML-файла при выводе его на экран, то есть игнорирует лишние пробелы, символы табуляции и символы конца строки. Если вы хотите, чтобы текст на экране выглядел так, как вы его ввели в HTML-документ, то воспользуйтесь тегом предварительного форматирования <PRE>. Текст должен находиться между тегами <PRE> и </PRE>.

Списки

Довольно часто требуется разместить на странице списки (перечни элементов). Списки бывают неупорядоченными и упорядоченными (по алфавиту или по цифрам). При отображении списков браузер выделяет их отступом от края страницы. Кроме того, списки могут быть вложенными.

Упорядоченные списки задаются тегом <OL>, а неупорядоченные — тегом <UL>. Оба эти тега парные, то есть контейнерные.

Для упорядоченных списков можно выбрать способ индексации. Это делается с помощью атрибута TYPE с аргументами: 1 (арабские цифры), A (прописные буквы), a (строчные буквы), I (римские цифры). Можно задать номер, с которого начинается нумерация элементов списка. Для этого служит атрибут START внутри тега <OL>.

Перед элементами списков следует поставить тег <LI>, чтобы индексация происходила автоматически. В этом теге можно использовать и вышеописанный атрибут TYPE.

В листинге П1.1 приведены примеры, показывающие различные способы создания списков (рис. П1.10).

Листинг П1.1. Различные способы создания списков

```
<HTM1>
<HEAO><T1T1E>Списки</TITLE></HEAD>
<BODY>
```

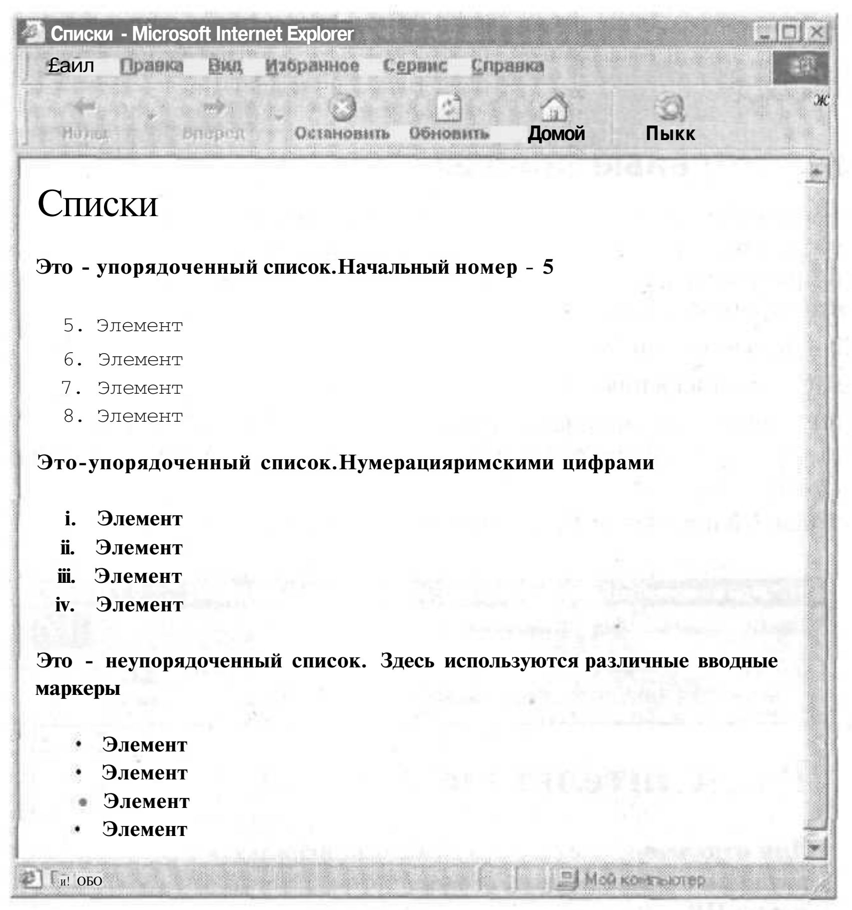


РИС. П1.10. Варианты списков

```
<H2>Списки</H2>
<P>Это - упорядоченный список. Начальный номер - 5<BR>
<OL TYPE=1 START=5>
<1Л>Элемент
<1л>Элемент
<1Л>Элемент
<LI>3neMeHT
</OL>
<P>Это - упорядоченный список. Нумерация римскими цифрами <BR>
<OL TYPE=i>
<11>Элемент
<1Л>Элемент
<1_1>Элемент
<LI>Элемент
</OL>
<P>Это - неупорядоченный список. Здесь используются вводные маркеры <BR>
<UL>
<11>Элемент
<11 >Элемент
<11>Элемент
```

```
<1_1>Элемент
</UL>
</BODY>
</HTML>
```

Разделительные полосы

При оформлении текста, чтобы отделить один раздел от другого, нередко используют разделительные полосы. Можно задать ширину, толщину и способ выравнивания разделительной полосы. Разделительная полоса задается тегом <HR>, внутри которого можно вставить атрибуты:

- SIZE — толщина в пикселах; •
- WIDTH — ширина в пикселах;
- ALIGN — способ выравнивания (принимает значения CENTER, LEFT или RIGHT); кроме того, можно использовать атрибут NOSHADE для создания сплошной черной полосы без тени.

Ниже приведен пример задания разделительных полос (рис. П1.11).

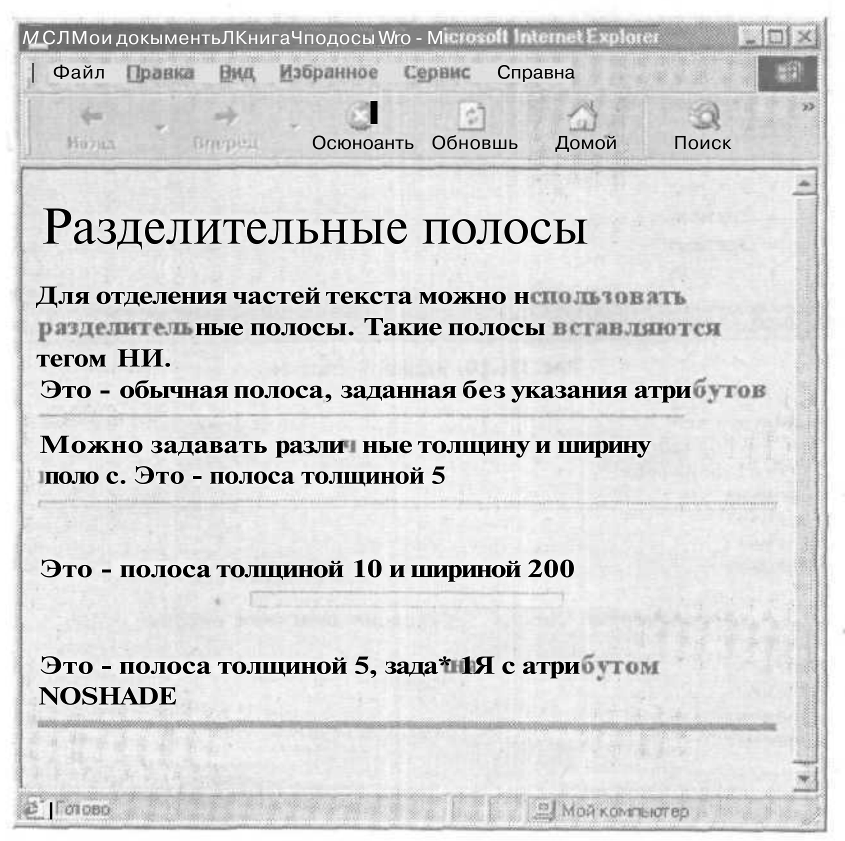


Рис. П1.11. Разделительные полосы

```
<HTML>
<BODY BGCOLOR="#E0E0E0">
<H1>Разделительные полосы<H1>
<H3>Для отделения частей текста можно использовать разделительные
полосы.Такие полосы вставляются тегом HR.<BR>
Это - обычная полоса, заданная без указания атрибутов
<HR>
Можно задавать различные толщину и ширину полос. Это - полоса толщиной 5
<HR SIZE=5>
<P>Это - полоса толщиной 10 и шириной 200
<HR SIZE=10 WIDTH=200>
<P>Это - полоса толщиной 5, заданная с атрибутом NOSHADE
<HR SIZE=5 NOSHADE>
<H3>
</BODY>
</HTML>
```

Задав равные небольшие значения для высоты и ширины, получим квадратик, который можно использовать в качестве маркера для разделения частей страницы или элементов списка. Однако в этом случае требуются специальные средства позиционирования элементов.

Бегущая строка

Internet Explorer поддерживает тег <MARQUEE>, который позволяет создать так называемую бегущую строку, то есть эффект прокручивания текста в заданном поле. Характеристики бегущей строки задаются следующими атрибутами:

- WIDTH — ширина поля бегущей строки в пикселах или процентах от ширины окна;
- HEIGHT — высота поля бегущей строки (в пикселах);
- HSPACE, VSPACE — интервалы по горизонтали и вертикали между текстом строки и краями ее поля (в пикселах);
- ALIGN — определяет положение текста бегущей строки в ее поле; возможные аргументы:
  - TOP (вверху);
  - BOTTOM (внизу);
  - MIDDLE (посередине);
- DIRECTION — определяет направление движения; возможные аргументы:
  - LEFT (справа налево);
  - RIGHT (слева направо);
- BEHAVIOR — характер движения строки; возможные аргументы:
  - SCROLL — текст появляется от одного края и скрывается за другим;
  - SLIDE — строка вытягивается из одного края поля и останавливается у другого края;
  - ALTERNATE — задает переменное направление движения, от одного края к другому, а затем обратно;

- LOOP — количество повторений текста в бегущей строке, задаваемое числом; если необходимо «бесконечное» количество повторений, то следует задать аргумент в виде ключевого слова INFINITY. Атрибут LOOP не влияет на поведение бегущей строки, если для атрибута BEHAVIOR заданы аргументы ALTERNATE или SLIDE;
- SCROLLAMOUNT — устанавливает длину в пикселах «прыжка» текста за один такт; при большом значении этого параметра текст движется рывками, а при малом — замедленно;
- SCROLLDELAY — определяет величину паузы между тактами перемещения текста в миллисекундах;
- BGCOLOR — устанавливает цвет поля бегущей строки, задаваемый шестнадцатеричным числом или именем.

Соотношения между длиной текста, размером шрифта и скоростью перемещения, при которых бегущая строка выглядит приемлемо, подбираются опытным путем.

В следующем примере бегущая строка «бесконечно» прокручивается на желтом поле шириной 75% от ширины окна браузера (рис. П1.12).

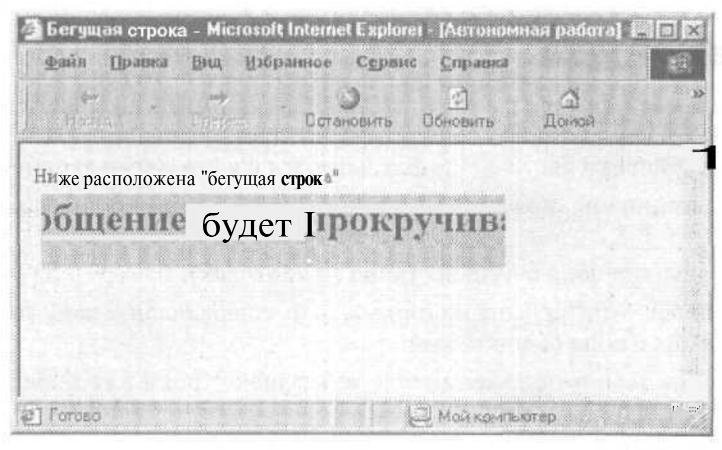


Рис. П1.12. Бегущая строка

```
<HTML>
   <HEAO><T1T1E>Бегущая строка</T1T1E ></HEAO>
<BODY>
Ниже расположена "бегущая строка"
<MARQUEE HEIGHT = 50 WIDTH=75% HSPACE=5 VSPACE = 5 ALIGN=TOP BGCOLOR=YELLOW
DIRECTION = LEFT LOOP=INFINITE BEHAVIOR=SCROLL SCROLLAMOUNT=5
SCROLLDELAY=100>
<H1> Это сообщение будет прокручиваться на экране справа налево</H1>
</MARQUEE>
</BODY>
</HTML>
```

Специальные и зарезервированные символы

При формировании HTML-документа может потребоваться ввести символы, которые воспринимаются браузером как служебные. Например, нельзя использо-

вать символы < и > для обозначения знаков «меньше» и «больше», так как они интерпретируются как символы тегов. Кавычки (") и амперсанд (&) также используются в языке HTML для служебных целей. Кроме того, все символы, которые можно набрать при нажатых клавишах управления, отличных от Shift, могут не воспроизводиться при просмотре HTML-документа, поскольку они зарезервированы.

ВНИМАНИЕ

Вместо зарезервированных символов в текст следует вставить их буквенные эквиваленты или коды ASCII. В качестве буквенного эквивалента используются соответствующие сокращения.

Буквенный эквивалент имеет формат:

`&буквенный_эквивалент;`

Например, < обозначает символ < (less then, меньше чем).

Цифровой эквивалент имеет формат:

&#код_ASCII;

Например, символ < можно вставить в документ как <.

Таким образом, эквиваленты заключаются между символами & и точкой с запятой. При этом в случае цифрового эквивалента перед ASCII-кодом символа следует поставить символ #. На рис. П1.13 представлена таблица эквивалентов для некоторых часто употребляемых зарезервированных символов. Данная таблица сформирована как HTML-документ. В листинге П1.2 приведено его содержание как пример использования специальных и зарезервированных символов.

Листинг П1.2. Код таблицы эквивалентов некоторых зарезервированных символов

```
<HTML>
<HEAD><TITLE>Специальные символы</TITLE></HEAD>
<H2>
<SAMP>
Симв.Код Эквивалент<BR>
<HR>
&#34;    34 quote<BR>
&#38;    38 amp<BR>
&#60;    60 lt<BR>
&#62;    62 gt<BR>
    160 nbsp<BR>
&#162;   162 cent<BR>
&#163;   163 pound<BR>
&#167;   167 sect<BR>
&#169;   169 copy<BR>
&#174;   174 reg<BR>
177;   177 plusmn<BR>
&#181;   181 micro<BR>
&#182;   182 para<BR>
&#188;   188 frac14<BR>
&#189;   189 frac12<BR>
&#190;   190 frac34<BR>
&#198;   198 AElig<BR>
&#216;   216 Oslash<BR>
&#230;   230 aelig<BR>
```

```
&#247;   247 divide<BR>
</H2>
</HTML>
```

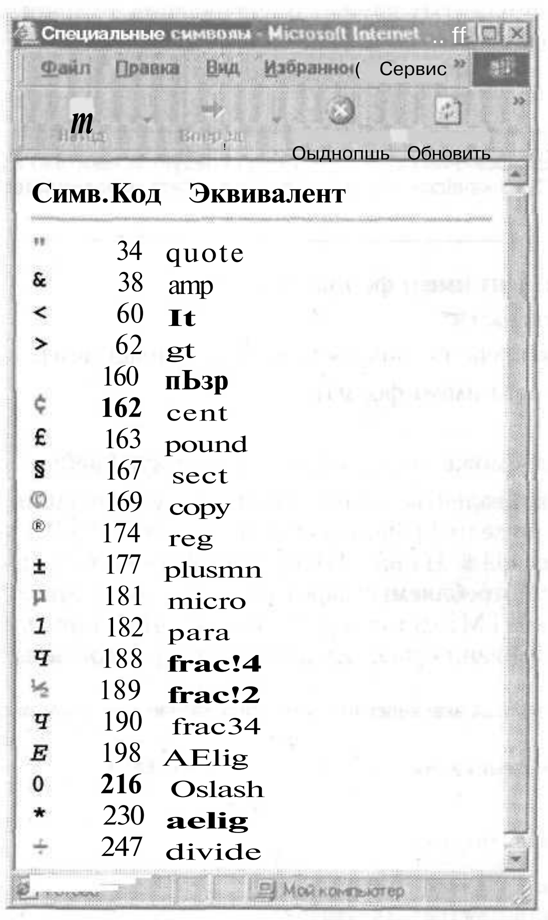


Рис. П1.13. Таблица эквивалентов для некоторых зарезервированных символов

Тег <SAMP> применен только для того, чтобы использовать моноширинный шрифт. Многократное использование символа ** ** (пробел) связано с тем, что браузер сокращает количество «лишних» пробелов, введенных с клавиатуры, а они были необходимы для задания достаточно большого расстояния между столбцами таблицы.

Графика на веб-страницах

В большинстве веб-страниц встречается графика. Она позволяет красочно и наглядно представить информацию. Во многих случаях лучше показать изображение, чем давать длинное текстовое описание.

Существует два способа размещения изображений на странице:

- вставка отдельных изображений;
- заполнение фона рисунком. »

В любом случае графический объект берется из файла.

Вставка изображений

Вставка на страницу изображения из файла графического формата производится с помощью тега <IMG> (от аш : image — изображение) с указанием адреса файла в качестве аргумента атрибута SRC:

```
<IMG 5KC="адрес_графического_файла">
```

Адрес графического файла — это либо URL-адрес, либо имя файла, возможно, с указанием пути.

Например, для показа графического файла logotip.jpg следует написать тег

```
<IMG SRC = "logotip . jpg">
```

Для увеличения скорости передачи изображения в теге <IMG> можно использовать атрибут (дополнительный параметр) LOWSRC, который принимает в качестве аргумента адрес графического файла. Вы можете создать д§а графических файла: один (например, пусть это файл logotip.jpg) содержит изображение, полученное с высоким разрешением, адругой (например, Logotip.gif) — рисунок, полученный с низким разрешением. Тогда тег

```
<IMG SRC="logotip.jpg" LOWSRC="logotip. gif">
```

предпишет браузеру сначала загрузить файл logotip.gif, а затем по мере загрузки страницы заменить его файлом logotip.jpg.

Другой способ ускорения загрузки графики заключается в задании размеров изображения с помощью атрибутов WIDTH (ширина) и HEIGHT (высота), измеряемых в пикселах. Если указать эти атрибуты, то браузер сначала выделит место под графику, подготовит макет документа, отобразит текст и только потом загрузит графику. Заметим, что браузер сжимает или растягивает изображение, встраивая его в рамки указанного размера. Пример указания размеров изображения:

```
<IMG SRC="logotip.gif" WIDTH=40 HEIGHT=20>
```

Здесь атрибут WIDTH задает ширину прямоугольной области, в которой будет размещено графическое изображение, а HEIGHT — высоту.

Графика обычно используется вместе с текстом, поэтому возникает задача выравнивания текста и изображения. Эта задача решается с помощью атрибута ALIGN тега<IMG> с применениемразличныхаргументов. Например, мы можем пожелать, чтобы текст обтекал рисунок справа или слева. Обычно изображение встраивается вплотную с текстом, что может быть некрасиво. Чтобы этого избежать, можно задать пустые поля вокруг иллюстрации. Поля создаются с помощью атрибутов VSPACE для верхнего и нижнего полей и HSPACE для боковых полей в теге <IMG>. Аргументы этих атрибутов указываются в виде чисел, определяющих размеры полей в пикселах. Для отмены обтекания графики текстом служит тег <BR CLEAR=...>.

Следующий тег задает обтекание графики из файла **logotip.jpg** справа (изображение будет находиться слева от текста):

.<**IMG** SRC="logotip.]'Pg" ALIGN=LEFT>

Если требуется расположить графический объект справа от текста, то необходимо атрибуту ALIGN присвоить аргумент RIGHT:

<IMGSRC="logotip.jPg"ALIGN = RIGHT>

Чтобы задать поля вокруг иллюстрации, следует написать тег вида:

<I MG SRC="logotip.jpg" ALIGN = LEFT HSPACE=20 VSPACE=10>

Здесь числа 20 и 10 определяют размеры полей.

Рассмотрим пример совместного использования графики и текстов. Вызовите Блокнот (текстовый редактор Notepad). Напишите в нем HTML-код с использованием рассмотренных выше тегов. Ниже приводится программа, выводящая некоторый текст и графику (рис. П1.14). В качестве графического файла можно использовать любой из имеющихся у вас файлов. Здесь используется файл **logotip.gif.**



Рис. П1.14. Совместное расположение текста и графики

```
<HTML>
<HEAD> <TITLE> Упражнение 1 </TITLE>
</HEAD>
<BODY BGCOLOR=" YELLOW" >
```

```
<IMG SRC="logotip.gif" ALIGN=LEFT>
<H1>Текст обтекает графику справа</H1>
Это - пример совместного использования текста и графики.<BR>
Текст программы HTML можно писать в любом текстовом редакторе.
При этом используются теги разметки текста.
<P>
Этот текст выводится с нового абзаца. Чтобы сделать это, мы использовали
специальный тег.
<P>
Попробуйте изменить размеры окна Вашего браузера. Обратите внимание, как
изменяется расположение текста.
</BODY>
</HTML>
```

Широкие возможности точного позиционирования изображений (как и других элементов) на странице предоставляют таблицы и стили. Эти элементы HTML будут рассмотрены позже. Например, вы можете определить таблицу без видимых рамок, а в ячейках этой таблицы разместить рисунки, тексты и другие элементы.

Фоновая графика

Чтобы украсить страницу, можно заполнить фон изображением из графического файла. Фоновое изображение — это файл, содержащий рисунок (желательно небольшого размера), который многократно выводится на экран, заполняя все окно.

Рисунок может представлять собой небольшой прямоугольник или же длинную узкую полоску (например, залитую градиентом).

Фоновая графика задается в теге <BODY> в начале HTML-документа, подобно тому как задается цвет фона. При этом используется атрибут BACKGROUND, значением которого является имя графического файла. Например, если мы хотим для фона взять файл fon.gif, то соответствующий тег будет выглядеть так:

```
<BODY BACKGROUND="fon.gif">
```

Для браузера Internet Explorer можно использовать дополнительный атрибут BGPROPERTIES=FIXED, запрещающий прокрутку фона в окне экрана. Например:

```
<BODY BACKGROUND="fon.gif" BGPROPERTIES = FIXED >
```

ВНИМАНИЕ

Следует иметь в **виду,** что существуют неграфические браузеры, а в графических браузерах пользователи могут отключать загрузку графики, поэтому можно задать и текстовое описание графики, встроенной в документ, то есть описать изображение или указать размер и формат графического файла.

При наличии текстового описания пользователь графического браузера сможет сам решить, стоит ли загружать тот или иной графический файл или лучше не тратить на это время. Подмена графики текстом осуществляется с помощью атрибута ALT в теге **<IMG>.**

Пример

```
<IMG SRC="logotip.gif" ALT="[Логотип, gif 18K]">
```

Ссылки

Ссылки (или гиперссылки) позволяют щелчком кнопкой мыши на выделенном тексте или изображении перейти к другому файлу или фрагменту страницы. Ссылки применяются в большинстве существующих страниц. Они могут быть текстовыми и графическими. Текстовые ссылки представляют собой выделенное слово или целую фразу. Выделение ссылки производится цветом или подчеркиванием, в зависимости от настройки браузера.

В языке HTML структуры текстовых и графических ссылок подобны друг другу. Все они задаются тегом <A **HREF**>, которому соответствует закрывающий тег </A>. В ссылке сначала указывается имя файла, на который она ссылается, а затем текст или имя графического файла, содержащего изображение ссылки. Кроме простых графических ссылок можно создать так называемую графическую карту ссылок: изображение с областями, щелчок на которых приводит к срабатыванию соответствующих ссылок.

Текстовые ссылки

Структура текстовой ссылки имеет следующий вид:

```
<A HKEP="адрес_ссылки"> текст_ссылки </A>
```

Обратите внимание, что тег ссылки имеет закрывающую часть </A>.

Например, следующий тег описывает ссылку на HTML-файл **Докум2.**htm, при этом ссылка на экране будет представлена текстом Щелкните здесь:

```
<A. HREF="4OKyM2.htin"> Щелкните здесь </A>
```

Отметим, что браузер не выводит на экран имя файла, к которому требуется перейти по ссылке, а лишь показывает текст, заключенный в теге между угловыми скобками > и <. Если же вы хотите, чтобы внешне ссылка выглядела как имя файла, на который она ссылается, то просто напишите его имя вместо текста.

Пример

```
<A HKEP="докум2.htm"> докум2.htm </A>
```

Можно ссылаться не только на другие файлы, но и на свой собственный файл. Поскольку настройки цвета в браузере у различных пользователей могут отличаться, возникает задача принудительно задавать цвета, чтобы ссылки были хорошо видны. Выше мы уже рассматривали, как задать цвет фона и текста. Это делается в теге <BODY> с помощью атрибута LINK для непрочитанной ссылки и **VLINK** — для прочитанной ссылки.

```
<BODY BGCOLOR="#303030" TEXT="BLUE" LINK="GREEN" VLINK="YELLOW">
```

Графические ссылки

Структура графической ссылки имеет вид:

```
<A НЯЕР="адрес_ссылки"> <IMG 5(?C="имя_графического_файла"> </A>
```

Например, следующий **тег** описывает ссылку на HTML-файл доку м2.И1т, при этом ссылка на экране будет представлена изображением из файла logotip.gif:

```
<A HREF="flOKyM2.htm"> <IMG SRC="logotip.gif"></A>
```

К графической ссылке можно добавить поясняющий текст, как это сделано ниже:

```
<A HREF="flOKyM2.htm"> <IMG SRC="logotip.gif"></A> Щелкните здесь
```

В рассмотренных выше графических ссылках одному изображению соответствовал один адрес ссылки. Однако имеется и другая возможность. Она заключается в том, чтобы одному изображению сопоставить несколько ссылок, привязав каждую из них к некоторой области изображения. Такие области называют горячими, а сам технологический прием — графической картой ссылок, или сегментированной графикой. Горячие области графической карты могут быть различной формы: прямоугольной, многоугольной или в виде окружности. Это очень удобный прием оформления группы ссылок, однако при выборе рисунка, служащего основой карты ссылок, следует стремиться к тому, чтобы границы горячих областей были хорошо очерченными и не пересекались. Кроме того, необходимо позаботиться о том, чтобы пользователь понял, что имеет дело с картой ссылок, а не просто с графическим фоном. Для этого можно использовать поясняющие тексты. При наведении на горячую область курсор мыши изменяет свою форму как и при использовании обычных ссылок. При щелчке на горячей области ее границы становятся видимыми.

Графическая карта задается с помощью нескольких тегов. Первым является тег **<MAP>** (карта) с атрибутом NAME для указания имени карты. Имя карты выбирается как имя переменной. Далее между тегами **<MAP>** и **</MAP>** следуют теги <AREA> (область) для задания горячих областей. Тег <AREA> имеет ряд атрибутов, описывающих собственно ссылку, а также форму и положение горячей области:

- HREF — строка, определяющая адрес ссылки;
- SHAPE — определяет форму области; принимает аргументы:
  - **"RECT"** (прямоугольник);
  - "POLYGONE" (многоугольник);
  - "CIRCLE" (круг);
- COORDS — координаты области, которые задаются **в** виде перечня чисел, разделенных запятыми; весь перечень заключается в кавычки (для прямоугольника задаются четыре числа — координаты верхнего левого и правого нижнего углов; для многоугольника задаются координаты каждого угла; для круга задаются три числа — координаты центра и радиус).

После закрывающего тега **</MAP>** следует указать тег, вставляющий изображение и реализующий привязку карты к нему, — это уже известный тег <IMG>, в котором помимо прочих возможных атрибутов используется атрибут связи с картой:

```
и$EMAP="#имя_карты"
```

В качестве имени карты указывается аргумент атрибута NAME тега <MAP>.

В нашем примере в основе графической карты ссылок находится топографическая карта некоторой местности. На ней мы определили прямоугольную и круглую горячие области, соответствующие двум населенным пунктам. При щелчке на горячей области будет выведен документ **северный.htm** или **южный.htm,** содержащий, например, описание соответствующего населенного пункта.

```
<HTML>
<HEAO><T1T1_E>Графическая карта</TITLE></HEAD>
```

```
<MAP NAME="map0">
<AREA HREF= "ceBepHbiti.htm" SHAPE="RECT" COORDS= "150,100,250,250">
<AREA HREF = "южный.htm" SHAPE = "CIRCLE" COORDS="150,380, 50">
</MAP>
<IMG SRC="Kapia.jpg" USEMAP="#map0" WIDTH=360 HEIGHT=480>
</HTML>
```

Графические карты ссылок обычно используются для создания красочных меню, а также в тех случаях, когда внешний вид страницы формируется с помощью графического редактора (например, Adobe Photoshop), в котором можно создать картинку со всеми необходимыми надписями и художественными элементами — это альтернатива использованию множества тегов, вставляющих различные элементы. Основная трудность заключается в определении координат горячих областей. Однако если известны ширина и высота всего изображения, то координаты горячих областей можно рассчитать хотя бы приблизительно, а затем уточнить опытным путем при отладке. Задача существенно упрощается при использовании такого средства проектирования веб-страниц, как FrontPage.

URL-адреса ссылок

В рассмотренных выше примерах в качестве адреса ссылки использовалось имя файла. В общем случае можно применять URL-адрес (Uniform Resource Locator - унифицированный указатель ресурса). Формат URL-адреса включает в себя тип сетевой службы (протокол связи), адрес сервера, путь поиска и имя файла.

Ниже перечислены URL наиболее популярных служб Интернета (табл. П1.3).

Таблица П1.3. URL наиболее популярных служб Интернета

Служба в Интернете	URL-адрес
World Wide Web	http://
FTP	ftp://
Mail	mailto:
Gopher	Gopher://
Телеконференции UseNet	news:

Если вы указываете адрес, начинающийся с http, тем самым вы обращаетесь к ресурсам, доступ к которым осуществляется по протоколу HTTP (Hyper Text Transfer Protocol — протокол передачи гипертекста). Этот протокол используется в качестве основного в Интернете при передаче информации, находящейся в HTML-документах.

Префикс адреса ftp означает, что следует использовать протокол передачи файлов (File Transfer Protocol — FTP). Этот протокол используется при передаче файлов-программ, имеющих расширение .exe. Он может использоваться при перемещении любых файлов с одного компьютера на другой. В частности, при перекачке файлов вашей страницы на сервер используется именно этот протокол. Протокол FTP обеспечивает высокую надежность передачи файлов. К примеру, если потерю до 10% обычной текстовой информации еще можно пережить, то при

передаче программы вообще не допускается потеря — неточно переданная программа просто не будет работать.

Если перед адресом ссылки указывается mailto, это означает, что следует использовать протокол передачи сообщений по электронной почте.

Gopher — служба (а значит, и протокол), предназначенная, в первую очередь, для работы неграфических браузеров. Она предоставляет систему доступа к информации, основанную на меню.

News — служба обеспечения телеконференций; это система типа доски объявлений, на которую вы можете поместить свое сообщение и на которой можно прочитать то, что там разместили другие участники телеконференции.

Ниже приведены примеры ссылок с применением URL-адресов:

```
<A HREF="ftp://ftp.admirat.ru/PUB/index.Шт">Бесплатный доступ</A>
<A HREF="mailto:dunaev@mail.admiral.ги">0тправить почту</A>
```

Пути поиска могут быть абсолютными и относительными. Абсолютный путь описывает местоположение файла начиная с самого высокого уровня и включает имена всех каталогов, ведущих к файлу. Ошибка в записи абсолютного пути (адреса) файла приводит к тому, что файл не будет найден. Относительный путь (адрес) описывает местоположение файла относительно места расположения текущего документа. Так, если вы указываете просто имя файла myfile.htm, это означает, что вы указываете относительный адрес. В данном случае браузер будет искать его в том же каталоге, где находится текущий документ. Если перед именем файла поставить ../ (например,. ./myfile.htm), то браузер будет искать файл в каталоге, находящемся на один уровень выше, чем тот, в котором находится текущий документ. Аналогично, если перед именем файла поставить ../../(например, ../.. /myfHe.htm), то браузер будет искать файл в каталоге на два уровня выше, чем текущий.

При создании ссылок вы можете указывать не только на конкретные документы и программы (то есть конкретные файлы, используя путь к ним), но и на папки (каталоги). Другими словами, адрес — это описание места расположения ресурса (единицы хранения информации). Он может быть точным или «приблизительным» (неполным). Вы можете ссылаться на папку, HTML-документ, документ, созданный каким-либо приложением (например, MS Word, MS Excel), или просто на текстовый файл с расширением .txt. Наконец, можно сослаться на файл программы с расширением .exe. Однако в последнем случае (по крайней мере, в Internet Explorer) сработает защита вашего компьютера. Появится окно с предупреждением и предложением возможных вариантов: запустить файл программы или сохранить его на диске — способ защиты от потенциальных вирусов. В этом случае вы сами решаете, что делать. Можно сразу запустить файл программы, а можно сохранить его на локальном диске, проверить с помощью антивирусной программы и только потом запустить, вылечить или уничтожить.

Ссылки в пределах одного документа

Иногда оказывается полезным организовать ссылки на разделы одного и того же документа. Например, на своей странице вы размещаете статью объемом в несколь-

ко десятков страниц как единый HTML-документ. Скорее всего, вам захочется сделать ссылки на предыдущие или последующие разделы этого документа. Речь идет о ссылках не на другие HTML-документы, а на определенные места того же самого документа. Такие ссылки еще называют закладками. Для них необходимы две вещи: якорь и собственно ссылка. Якорь определяет место в документе, к которому происходит переход по ссылке. Ссылка использует имя якоря вместо имени (адреса) файла.

Формат якоря:

<A MME="имя_якоря"> текст_на_экране</A>

Формат ссылки:

<A HKEP"#имя_якоря"> текст_на_котором_щелкать</A>

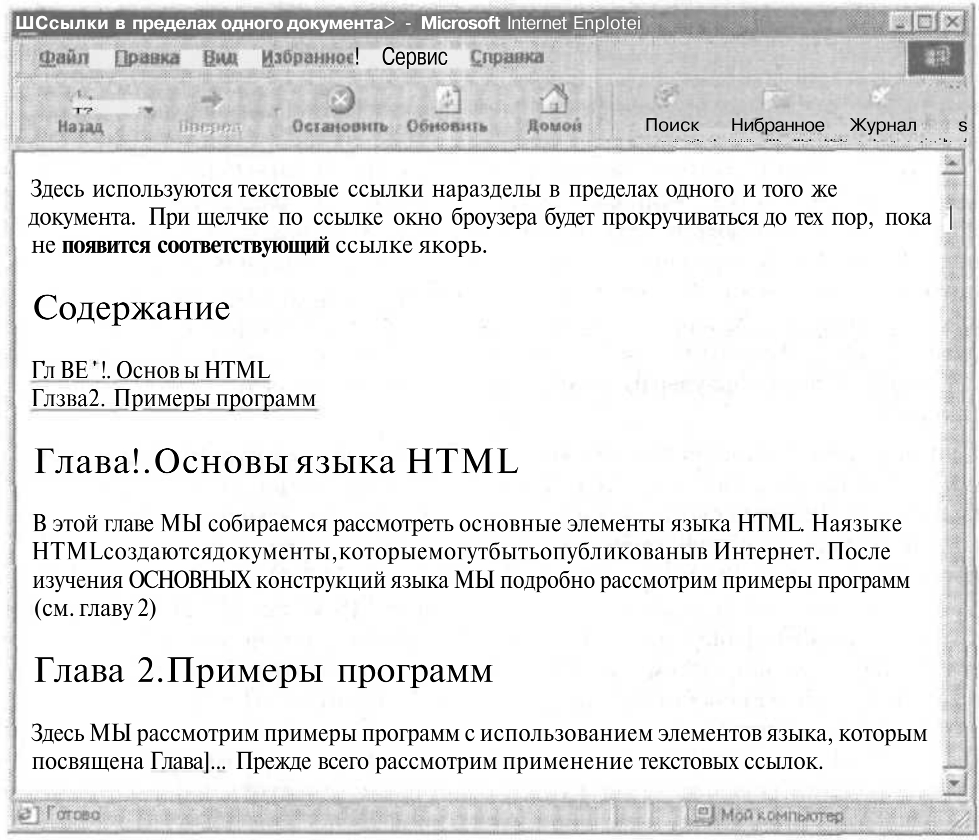


Рис. П1.15. Текстовые ссылки на разделы в пределах одного и того же документа

В листинге П1.3. приводится программа, демонстрирующая применение ссылок в пределах одного документа и вид этого документа в окне браузера (рис. П1.15). Мы используем ссылки при создании оглавления документа, а также ссылки из текста одних глав на другие главы.

Листинг П1.3. Код программы, демонстрирующей применение ссылок в пределах одного документа

```
<HTML>
<HEAO><T1T1_E>Ссылки в пределах одного документа</T1T1_E></HEAO>
<BOOУ>Здесь используются текстовые ссылки на разделы в пределах одного и
того же документа. При щелчке на ссылке окно браузера будет
прокручиваться до тех пор, пока не появится соответствующий ссылке
якорь.
<P>
<H2>Содержание</H2>
<p><p><A НЖЕР "#Глава1">Глава1. Основы HTML</A>
<BR><A НКЕР="#Глава2">Глава2. Примеры программ</A>
<p><p>
<H2><A МАМЕ = "Глава1">Глава1 , </A>Основы языка HTML</H2>.
В этой главе мы собираемся рассмотреть основные элементы языка HTML. На
языке HTML создаются документы, которые могут быть опубликованы в
Интернете. После изучения основных конструкций языка мы подробно
рассмотрим примеры программ (<A НКЕР="#Глава2">см. главу 2</A>) .
<P>
<H2><A МАМЕ="Глава2">Глава 2.</A>Примеры программ</H2>
Здесь мы рассмотрим примеры программ с использованием элементов языка,
которым посвящена
<A HREF="#[7iaBal">rnaBal.</A>
Прежде всего рассмотрим применение текстовых ссылок.
</BODY>
</HTML>
```

Таблицы

Довольно часто на веб-страницах встречаются таблицы. Простейший пример табличной организации данных— прайс-лист. Однако таблицы можно использовать просто как способ форматирования текста, поскольку таблица в HTML-документе может не содержать всех или некоторых разграничительных линий (сетки). Заметим, что в большинстве случаев элементы текста позиционируются как табличные.

Идея использования таблиц в качестве средства позиционирования проста. Вы мысленно разбиваете окно на ячейки (клетки) и решаете, какой элемент страницы в какую ячейку поместить, — это хороший способ избавиться от «естественного» расположения элементов друг за другом, которое осуществляет браузер по умолчанию. Далее будет рассмотрен более «свободный» способ позиционирования элементов, позволяющий помещать элементы в любое место окна браузера. Он основан на каскадных таблицах стилей. Однако, как правило, на практике для позиционирования элементов вполне достаточно использовать простую схему таблицы.

Таблица представляет собой особую часть HTML-документа. Данные в ней организованы в виде прямоугольной сетки, которая состоит из вертикальных столбцов (колонок) и горизонтальных рядов (строк). Каждая клетка таблицы называется ячейкой. Ячейки могут содержать текст, грасрику и даже другую таблицу.

Таблицы состоят из трех основных частей: названия таблицы, заголовков столбцов и ячеек. Таблица заполняется слева направо, ячейка за ячейкой, начиная с ле-

вого верхнего угла и заканчивая правым нижним углом. Каждая ячейка должна быть заполнена (для создания пустых ячеек используются пробелы).

Описание таблицы в HTML-документе начинается с тега <TABLE> и заканчивается тегом </TABlE>. Если вы хотите, чтобы таблица имела видимую рамку (границы), используйте атрибут BORDER, например:

```
<TABLE BORDER>
```

Атрибут BORDER может принимать аргумент (число), определяющий ширину рамки, например:

```
<TABLE BORDER=10>
```

Для задания названия таблицы используется тег <САРТЮМ> с атрибутом выравнивания ALIGN, который может принимать значение TOP или BOTTOM (расположение названия вверху или внизу таблицы соответственно), например:

```
<CAPTION ALIGN=TOP> Моя таблица </CAPTION>
```

Каждый ряд ячеек начинается с тега <TR> и заканчивается тегом </TR>.

Если ряд должен содержать заголовки столбцов таблицы, то используются теги <TH> и </TH>. Если в ячейках должны размещаться данные, то используются теги <TD> и </TD>.

Теги заголовков и данныхдолжны располагаться между тегами рядов <TR> и </TR>. Проще говоря, вы сначала определяете таблицу (тег <TABLE>), а затем внутри этого определения задаете строки (тег <TR>) и содержимое ячеек (тег <TD> — для данных, тег<TH> — для названий заголовков столбцов).

Приведем пример описания простой таблицы, встроенной в некоторый текст. Обратите внимание на порядок заполнения ячеек таблицы. Сначала определяется строка заголовков столбцов таблицы. Затем аналогичным образом создаются строки данных, при этом указывается, какие данные должны располагаться в ячейках таблицы. Строка заголовков столбцов задается так же, как и строка данных. Отличие лишь в том, что при создании строки заголовков внутри тега <TR> используется тег <TH>, а при создании строки данных — тег <TD>.

```
<HTML>
<HEAO><T1T1_E>Таблицы</T1T1_E></HEAO>
<BODY>
<H2>Основные элементы таблиц</H2>
<P>
```

Ниже приведен пример простой таблицы (рис. П1.16).

```
<P>
<TABLE BORDER>
<CAPTION ALIGN=TOP>CnHCOK сотрудников</САРТЮМ>
<TR>
<TH>Имя</TH><TH>Должность</TH><TH>Оклад</TH>
</TR>
<TR>
<TD>MBaH</TD><TD>4npeKTop</TD><TD>1000</TD>
</TR>
<TR>
<TD>neTp</TD><TD>ByxranTep</TD><TD>800</TD>
</TR>
<TR>
<TD><t>eflop</TD><TD>Pa6o4MPi</TD><TD>50</TD>
```

```
</TR>
</TABLE>
</BODY>
</HTML>
```

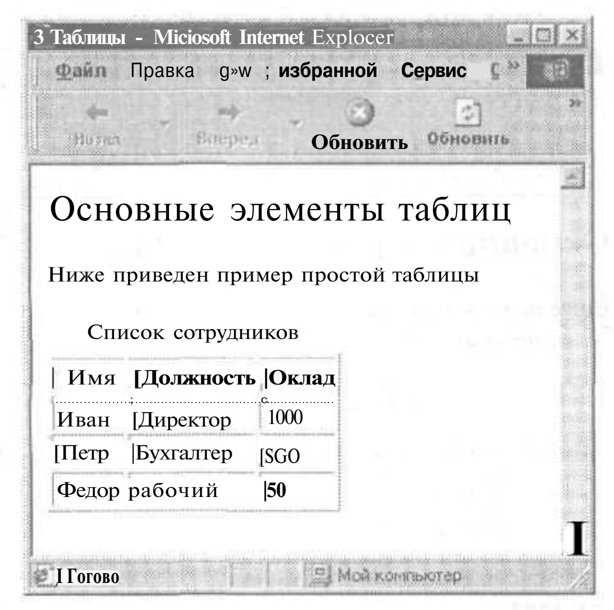


Рис. П1.16. Пример простой таблицы

Несколько ячеек можно объединить в одну как по горизонтали, так и по вертикали. Объединение по столбцу применяют в том случае, когда необходимо, чтобы соседние столбцы имели общий заголовок. Объединение по рядам делается тогда, когда содержимое нескольких ячеек подряд одинаково. Для объединения ячеек используются теги COLSPAN и ROWSPAN (по столбцам и по рядам соответственно). Аргументами этих атрибутов будет количество объединяемых столбцов или рядов. Вот пример объединения ячеек:

```
<HTML>
<HEAD><TITLE>Ta6nnubi</TITLE></HEAD>
<BODY>
<H2>Основные элементы таблиц</H2>
<P>
```

Ниже приведен пример простой таблицы с объединенными ячейками (рис. П1.17).

```
<P>
<TABLE BORDEFO
<CAPTION AlЛCM=TOP>Список сотрудников</САРТЮМ>
<TR><TH C015PAM=2>ИМЯ И должность</TH><TH>Оклад</TH>
</TR>
<TR><TD>MBaH</TD><TD>4HpeKTOp</TD><TD>1000</TD>
</TR>
<TR><TD>rieTp</TD><TD>ByxranTep</TD><TD>800</TD>
</TR>
<TЯ><TO>Федор</TO><TO ROWSPAN=2>Pa6o4iifi</TD><TD>50</TD>
</TR>
```

```
<TR><TD>BacnnMU</TD><TD>70</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

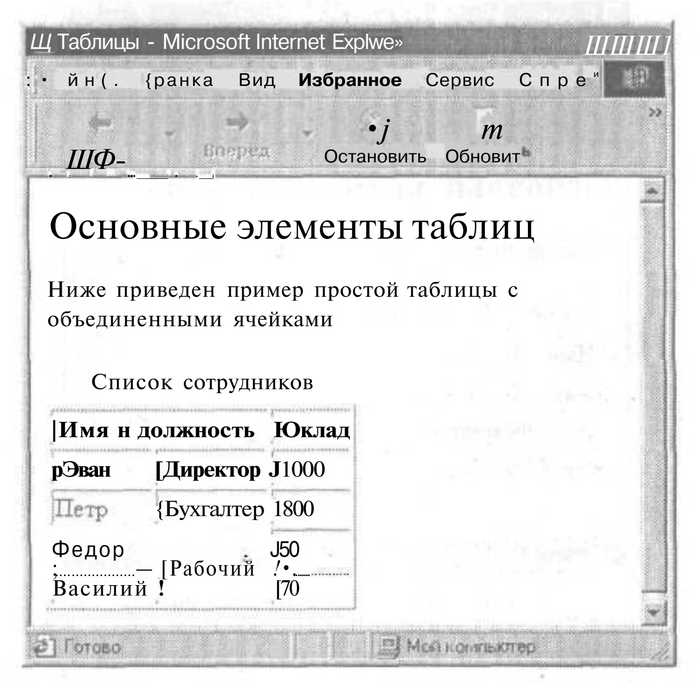


Рис. П1.17. Пример простой таблицы с объединенными ячейками

Можно управлять шириной как всей таблицы, так и каждой отдельной ячейки. Ширину всей таблицы можно задать как в пикселах, так и в процентах от ширины окна. Ширину ячеек можно задать также двумя способами: в пикселах и в процентах от ширины таблицы. Следует иметь в виду, что ячейки в одном столбце имеют одинаковую ширину, а ячейки одного ряда — разную ширину. Ширина таблицы задается атрибутом WIDTH в теге <TABl_E>, ширина ячейки указывается тем же атрибутом в теге <TH> или <TD>.

```
<HTML>
<HEAD><TITLE>Ta6nnubi</TITLE></HEAD>
<BODY>
<H2>Основные элементы таблиц</H2>
<P>
```

Ниже приведен пример таблицы с объединенными ячейками и заданными размерами (рис. П1.18).

```
<P>
<TABLE BORDER WIDTH=400>
<CAPTION A1ЛCЫ=TOP>Список сотрудников</CAPTЮM>
<TR>
<TH COLSPAN=2 ИЮTH=75%>Имя и должность</TH> <TH>Оклад</TH>
</TR>
```

```
<TR>
<TD>MBaH</TD><TD>AMpeKTop</TD><TD>1000</TD>
</TR>
<TR>
<TD>fleTp</TD><TD>ByxranTep</TD><TD>800</TD>
</TR>
<TR>
<TO>Федор</TO><TO  ROWSPAN=2>Pa6o4Mii</TD><TD>50</TD>
</TR>
<TR>
<TD>BackmMii</TD><TD>70</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

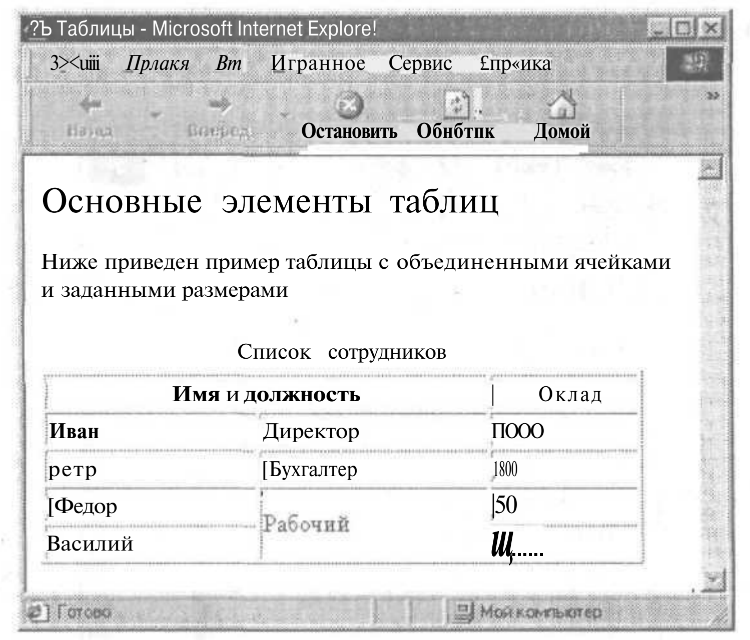


Рис. П1.18. Пример таблицы с объединенными ячейками и заданными размерами

Текст и графику внутри ячеек таблиц можно выравнивать. Горизонтальное и вертикальное выравнивание содержимого рядов задается с помощью атрибута ALIGN в **теге** <TR>. Выравнивание в отдельных ячейках выполняет атрибут ALIGN в тегах <TH> и <TD>. Атрибут ALIGN может принимать аргументы LEFT (по левому краю), RIGHT (по правому краю) и CENTER (по центру).

Примеры использования атрибута ALIGN:

```
<TR  ALIGN=LEFT>
<тн wiDTH=25% ALIGN=CENTER>HMH и должность</тн>
```

Для определения дизайна рамок таблицы в браузере Internet Explorer можно использовать атрибут FRAME тега <TABLE>. Далее перечислены возможные аргументы атрибута FRAME, указывающие способы изображения рамки:

- BOX—все четыре стороны рамки;
- ABOVE — только верхняя часть рамки;
- BELOW — только нижняя часть рамки;
- HSIDES — горизонтальные части рамки сверху и снизу;
- VSIDES — только левая и правая вертикальные части рамки;
- LHS — только левая часть рамки;
- RHS—только правая часть рамки;
- VOID — не изображать внешнюю рамку.

Разделительные линии между столбцами и рядами таблицы описываются атрибутом RULES в теге <TABLE>. Атрибут RULES может принимать следующие значения, задающие способ изображения разделительных линий:

- ALL — все вертикальные и горизонтальные линии;
- ROWS — только горизонтальные линии между рядами;

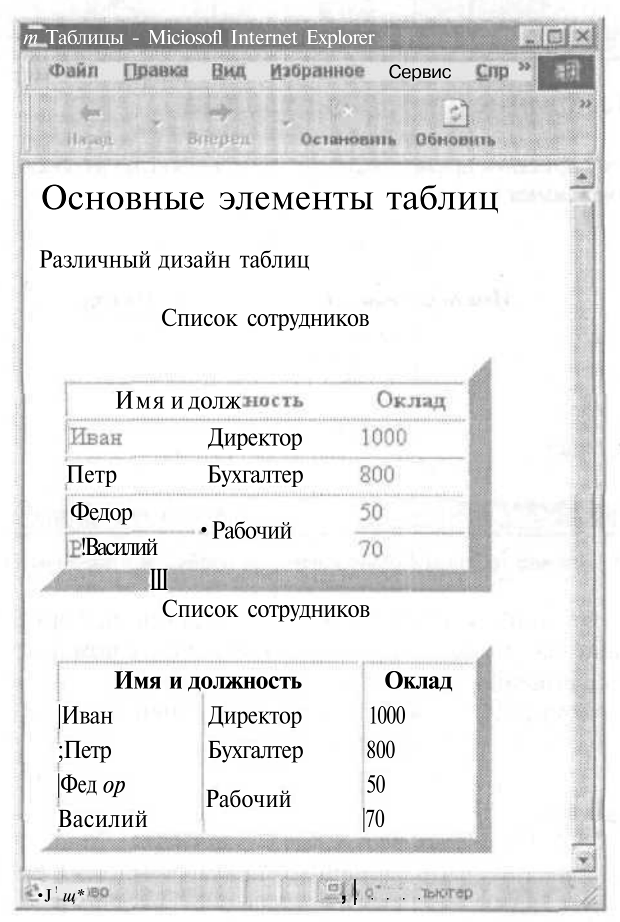


Рис. П1.19. Пример различного дизайна таблиц

- COLS — только вертикальные линии между столбцами;
- NONE — не изображать разделительные линии.

В листинге П1.4 приведен пример управления дизайном таблиц (рис. П1.19).

Листинг П1.4. Пример управления дизайном таблиц

```
<HTML>
<HEAO><T1T1_E>Таблицы</TITLE></HEAD>
<BODY>
<H2>Основные элементы таблиц</H2>
<P>Различный дизайн таблиц
<P>
<TABLE BORDER=15 FRAME=BOX RULES=ROWS WIDTH=300>
<CAPTION ALIGN=TOP>CnncoK сотрудников</CAPTЮM>
<TR>
<TH COLSPAN=2>ИМЯ И должность</TH><TH>Оклад</TH>
</TR>
<TR>
<TD>MBaH</TD><TD>AMpeKTop</TD><TD>1000</TD>
</TR>
<TR>
<TD>neTp</TD><TD>ByxranTep</TD><TD>800</TD>
</TR>
<TR>
<TO>Федор</TO><TO ROWSPAN=2>Pa6o4M</TD><TD>50</TD>
</TR>
<TR>
<TD>BacMnMii</TD><TD>70</TD>
</TR>
</TABLE>
<TABLE BORDER=10 FRAME=BOX RULES=COLS WIDTH=300>

<CAPTION ALIGN=TOP>CniicoK сотрудников</CAPTЮM>
<TR>
<TH C01_5PAM=2>Имя и должность</TH><TH>Оклад</TH>
</TR>
<TR>
<TO>Иван</TO><TO>Директор</TD><TD>1000</TD>
</TR>
<TR>
<TD>rieTp</TD><TD>ByxranTep</TD><TD>800</TD>
</TR>
<TR>
<TO>Федор</TO><TO ROWSPAN = 2>Pa6o4MU</TD><TD>50</TD>
</TR>
<TR>
<TD>BacnnnPi</TD><TD>70</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

В следующем примере используем таблицу без рамок для размещения различных элементов на странице. В ячейки помещены текстовые ссылки, изображения и число 1000, а некоторые ячейки остались пустыми (рис. П 1.20). Для задания нужного расстояния между элементами следует использовать атрибуты выравнивания. Пока еще нет другого способа разместить несколько элементов в одном ряду, кроме

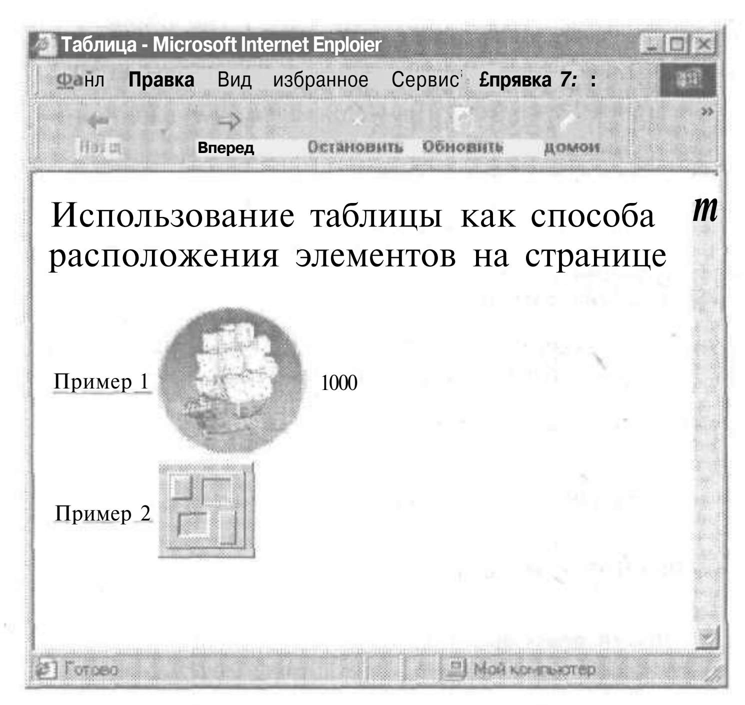


Рис. П1.20. Использование таблицы для расположения элементов на странице

использования таблиц. В следующем разделе, посвященном каскадным таблицам стилей, будет рассмотрен другой способ позиционирования элементов. В приведенном ниже примере просто задана таблица, но не указаны ее размеры и способ выравнивания для того, чтобы показать: можно располагать элементы как по горизонтали, так и по вертикали в пределах некоторой области. При этом только опытный программист на языке HTML может догадаться по внешнему виду страницы, что для ее создания была использована таблица. Освойте этот прием позиционирования, если хотите располагать на веб-странице много графических элементов и ссылок. Это позволит сэкономить место и достаточно эффектно разместить материал в окне браузера.

```
<HTML>
<HEAO><T1T1E>Таблица< /T1T1E ><HEAO>
<BODY>
<H2>
Использование таблицы как способа расположения элементов на странице
</H2>
<TABLE>
<TR>
<TD><A HREF = "npnMepl.Ь1т">Пример 1</A></TD>
<TD><IMAGE SRC="logotip.gif" WIDTH=100 HEIGHT=100></TD>
<TD>1000</TD>
</TR>
<TR>
<TD><A HREF="npnMep2.htm">npHMep 2</A></TD>
<TD><IMAGE SRC="квадрат.bmp"></TD>
</TR>
```

```
</TABLE>
</BODY>
</HTML>
```

Цветовое оформление таблиц производится с помощью атрибута BGCOLOR, который принимает в качестве аргумента цвет в виде имени или шестнадцатеричного числа. Если нужно установить цвет для всей таблицы, то атрибут BGCOLOR вставляется в тег <TABLE>. Для изменения цвета только одного ряда этот атрибут вставляется в тег <TR>. Задание цвета отдельной ячейки обеспечивается атрибутом BGCOLOR внутри тега <TD>.

Если в таблице имеются рамки, то можно указать и их цвет с помощью атрибутов:

- BORDERCOLOR - цвет всей рамки;
- BORDERCOLORLIGHT - цвет светлой части рамки;
- BORDERCOLORDARK - цвет темной части рамки.

Эти атрибуты вставляются в тег <TABLE>. Чтобы они действительно работали, необходимо наличие еще и атрибута BORDER, задающего ширину рамки. Ниже приводится пример использования атрибута цвета в различных тегах таблицы:

```
<TABLE BGCOLOR="#F0F0F0" BORDER=10 BORDERCOLOR="#808080"
BORDERCOLORLIGHT="#707070" BORDERCOLORDARK="#202020">
<TR BGCOLOR="BLUE">
<TH>Имя</TH><TH>Адрес</TH>
</TR>
<TR>
<TD BGCOLOR="YELLOW">MBaH</TD><TD>r.Санкт-Петербург</T0>
</TR>
</TABLE>
```

Задать шрифт для текстов внутри ячеек таблицы можно с помощью тегов заголовков и физических стилей внутри тегов <TD>:

```
<TD BGCOLOR="YELLOW"><H2>MBaH</H2></TD>
```

Теперь вы имеете общее представление о том, зачем нужны таблицы и как их создавать. Заметим, что таблицы удобно готовить в таких редакторах, как FrontPage и Dreamweaver. С их помощью визуальными средствами можно начертить таблицу и вставить в ее ячейки необходимые данные. Для этих операций редактор Dreamweaver даже удобнее, чем FrontPage.

Стили

Существует еще один прием позиционирования элементов для достижения визуальных эффектов, которые могут украсить страницу и привлечь к ней внимание. Этот прием основан на определении пользовательских стилей и задании таблицы стилей. Таблица стилей — это просто некоторая структура описания свойств элемента. Не ищите здесь прямоугольной сетки. Если таблица стилей задана, то различные документы могут просто ссылаться на эту таблицу и не содержать большое количество атрибутов в тегах форматирования типа <Hl>, <FACE> и т. п. Таблицы стилей (так называемые Cascading Style Sheets — CSS) содержат описание формата части или всего текста, координаты расположения элементов и другие параметры. Задание стиля обеспечивается с помощью и тега <STYLE>, и атрибута STYLE.

Используя стили, можно позиционировать элементы страницы (например, тексты и графику), указав координаты их положения. И это, пожалуй, самое важное, что дают стили. Кроме того, таблицы стилей часто применяются при создании динамических страниц.

С этого момента способ изложения материала несколько изменится. Теперь мы больше будем рассказывать об идеях, иллюстрируя их примерами, и меньше уделять внимание полноте определений. При этом все примеры программ будут, как и прежде, работоспособными. Даже если вы не до конца поймете теоретический материал, вы все равно можете использовать приведенные ниже примеры в своих разработках. Для этого следует просто скопировать тексты или отдельные их фрагменты в ваш собственный HTML-документ и адаптировать к своей конкретной задаче путем коррекции отдельных параметров.

Теги <STYLE> и </STYLE> используются внутри тегов заголовка файла <HEAD> и </HEAD>, а атрибут STYLE — в теге заголовка раздела (<H1>, <H2>,...,<H6>), а также в теге <BODY>, в теге выделения фрагмента <DIV> и многих других.

Применение тега <STYLE>:

```
<HEAD>
<STYLE>
Тег {свойство!: значение! ; свойство2 : значение2,..., свойством:
значением }
</STYLE>
</HEAD>
```

Здесь в фигурных скобках через точку с запятой перечисляются свойства и их значения, причем между свойством и его значением ставится двоеточие. При этом тег, указываемый перед описанием в фигурных скобках, пишется без угловых скобок.

В следующем примере определяются стили заголовков первого и второго уровней путем использования тега STYLE:

```
<HTML>
<HEAD>
<STYLE>
HI {Font-size:48pt;Color:RED}
H2 {Font-size:20pt;Color:BLUE}
</5TYLE>
</HEAD>
<BODY>
<H1>Это стиль HI. Цвет - красный</H1>
<P>
<H2>Это стиль H2. Цвет - синий</H2>
<P>
Это - обычный стиль по умолчанию
</BODY>
</HTML>
```

Мы изменили стили HI и H2, принятые по умолчанию: назначили размеры (48 и 30 точек) и цвета (красный и синий) для всех текстов, которые окажутся внутри этих тегов. Существуют и другие свойства. Например, свойство FONT-FAMILY: HELVETICA задает гарнитуру шрифта HELVETICA.

Обратите внимание на то, что внутри тега <STYLE> указываются определения стилей тегов, которые записываются без угловых скобок.

Можно определить стиль для тега BODY. Тогда весь текст, находящийся между тегами <BODY> и </BODY>, будет автоматически отформатирован в соответствии с перечисленными свойствами.

Если мы хотим задать один и тот же стиль сразу для нескольких тегов, то перед определением стиля (всего, что заключено в фигурные скобки) можно указать перечень тегов (без угловых скобок), разделенных запятыми. В следующем примере задается одинаковый стиль для заголовков первого и второго уровней:

```
<HTML>
<HEAD>
<STYLE>
HI, H2 {font-size:48pt;co\or:RED}
</STYLE>
</HEAD>
<BODY>
<H1>Это стиль HI. Цвет - красный</H1>
<P>
<H2>Это стиль H2, такой же, что и HI. Цвет - красный</H2>
<P>
Это - обычный стиль по умолчанию
</BODY>
</HTML>
```

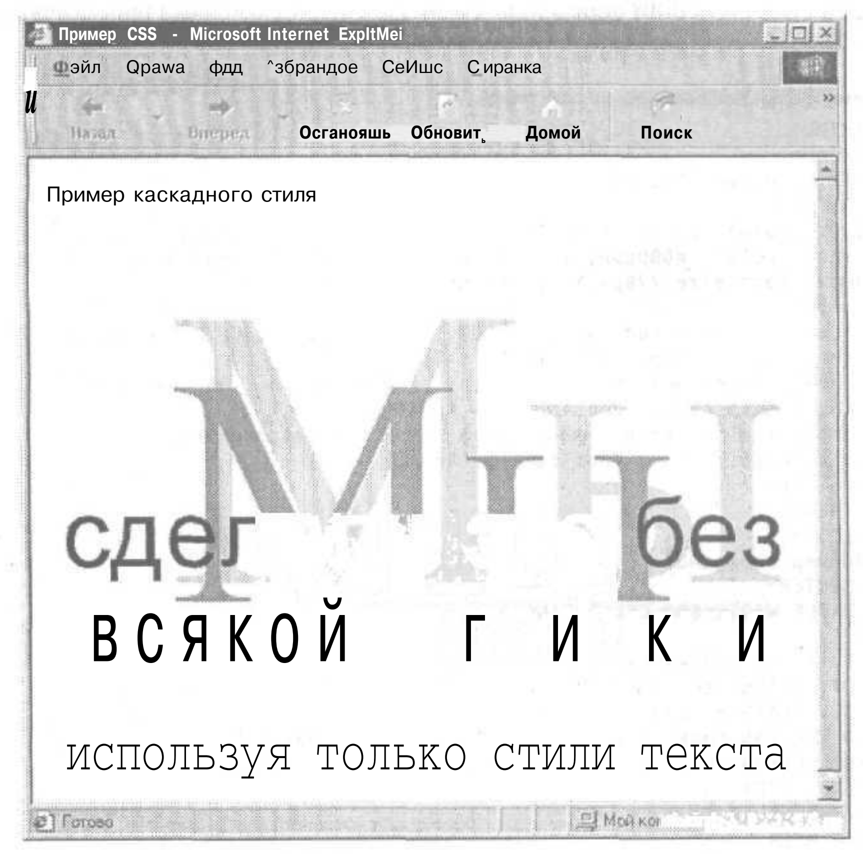


Рис. П1.21. Пример каскадного стиля

Мы можем создать таблицы стилей, закрепив за ними имя. Это имя задается как обычное имя, но с точкой в качестве первого символа. Тогда в тегах мы можем обращаться к этой таблице по ее имени, используя атрибут C1_A5$=имя_стиля, где имя стиля употребляется уже без точки.

Например, мы можем задать стиль так:

```
<STYLE>
mystyle  {color:black; font-family: Arial}
</5TYLE>
```

А стиль заголовка второго уровня можно задать где-нибудь в тексте программы таким образом:

```
<H2 CLASS=mystyle>
```

Рассмотрим пример (листинг П 1.5), создающий эффект, которого без задания стиля можно было бы достичь только с помощью графики (рис. П1.21). Мы накладываем тексты друг на друга. Эта возможность далее будет использована для создания эффекта выпуклого текста (так называемого SD-эффекта). Кроме того, возможность наложения (частичного перекрытия) фрагментов страницы часто используется в дизайне реальных страниц. В данном примере применяется тег <DIV> для выделения фрагмента HTML-документа. Обратите внимание, как внутри определения стиля тега BODY определяются другие стили: с именами тень, основа, слой! и слой2. Это и есть каскадные таблицы стилей.

Листинг П1.5. Создание каскадного стиля

```
<HTML>
<HEAD>
<T1T1E>Пример CSS</TITLE>
<STILE>
BODY {color: black; font-size:16px; font-family: Arial}
 .тень {color: #DBDBDB; text-align:right; weight: medium; margin-top:
30px; font-size:270px;line-height: 270px;
font-family: Times}
 .основа {color: red; weight: 900; margin-top: -230px; font-size:220px;
line-height: 250px; font-family: Times}
 .слой! {color: black; margin-top: -130px; weight: medium; ont-size:65px;
line-height: 65px; font-family: Arial}
 .слой2 {color: green; margin-top: 30px; weight: medium; font-size: 35px;
line-height: 45px; font-family: Arial}
</STYLE>
</HEAD>
<BODY>
Пример каскадного стиля
<CENTER>
<TABLE WIDTH=500 CELLPADING=0 CELLSPACING=0 BORDER=0>
<TR>
<TD ALIGN=CENTER VALIGN=TOP>
<DIV CLASS=TeHb>Mbi</DIV>
<DIV CLASS=ocHOBa>Mbi</DIV>
<DIV C1A55=слой1>сделали это без всякой графики</01У>
<DIV C1_A55=слой2>используя только стили текста</01У>
</TD></TR>
</TABLE>
</CENTER>
```

```
</BODY>
</HTML>
```

В приведенном выше примере использованы тег <DIV> и атрибут CLASS.

Тег <DIV> применяется для **задания** части страницы (фрагмента документа). Он ничего не форматирует, а лишь помечает фрагмент текста, который рассматривается как единый объект. Атрибут CLASS позволяет сослаться на таблицу стилей и тем самым задать стиль представления текста, расположенного между тегами <DIV CLASS.. .> и </DIV>. Обратите внимание на то, как в таблице стилей определяется стиль: набору свойств в фигурных скобках присваивается имя, перед которым ставится точка. В дальнейшем идут ссылки на эти имена с помощью атрибута CLASS для применения ранее определенного стиля. Идея проста: сначала определяется что-то, а затем используется это определение путем ссылки на него.

В этом примере тексты определяются как бы в слоях, которые накладываются друг на друга. Так, сначала выводится слой тень, затем на него накладывается слой основа, апотом — слой! ислой2. Очередность, в которой слои накладываются друг на друга, задает порядок следования фрагментов текста, помеченных тегом <DIV>. Собственно перекрытие слоев обеспечивается применением отрицательных значений свойства margin-top (отступ сверху).

В рассмотренном выше примере были использованы следующие свойства:

- margin-top — отступ сверху;
- color — цвет;
- font-size — размер шрифта;
- font-family — семейство шрифтов;
- weight — степень жирности шрифта;
- line-height — высота строки.

Используя отрицательные значения свойства margin-top, можно наложить один текст на другой.

Кроме рассмотренных выше способов задания стилей можно применять атрибут STYLE, вставляемый в теги заголовков, **абзаца <P>, тела<BOOY>, <DIV>** и **<IMG>. В** этом случае стиль задается в следующем формате:

```
5TY1E="описание_свойств_стиля"
```

Описание свойств стиля заключается в кавычки и содержит свойства и их значения, перечисленные через точку с запятой, как это делалось при использовании тега **<STYLE>.**

Атрибут STYLE обычно применяется в тех случаях, когда необходимо задать стиль элемента по месту его появления в программе. Вот пример использования атрибута STYLE для форматирования заголовка второго уровня:

```
- <H2 STYLE="font-size:48: font-family:Aria1">Некоторый текст</H2>
```

Позиционирование элементов

Выше мы говорили о том, что позиционировать элементы страницы можно и путем использования таблиц. Но это же можно делать и с помощью стилей. Среди параметров стиля имеются специальные свойства для позиционирования:

- left — для задания расстояния в пикселах от левого края окна(x-координата);
- top — для задания расстояния в пикселах от верхнего края окна (y-координата);
- **z-index** — для указания порядка, в котором элементы будут перекрывать друг друга; это новое измерение, элементы с большим z-индексом будут появляться над элементами с меньшим z-индексом.

Конечно, при использовании этих трех свойств не создается эффекта трехмерного пространства, но это уже нечто большее, чем двухмерная плоскость. В этом случае говорят о 2,5-мерном пространстве.

Кроме свойств-координат нам понадобится свойство **position,** которое в сочетании со свойствами left и top позволяет устанавливать элементы в определенные позиции окна. Свойство position может принимать три значения.

- absolute — заданные свойства Left и top устанавливают элемент в место с координатами x и y относительно верхнего левого угла контейнера (объекта, содержащего данный элемент). Если они определены для элемента вне контейнера, то началом отсчета координат будет верхний левый угол страницы. Заметим, что положение элемента не зависит от положения его тега внутри HTML-документа.
- relative — элемент будет установлен в соответствии с тем, в каком месте исходного текста он находится; это значение установлено по умолчанию. Например, если элемент находится в трех строках от начала **его** выделения в тексте документа, то по умолчанию считается, что свойство позиционирования имеет значение relative, а свойства координат left и top — нулевые значения. Ненулевые значения свойств left и top сдвигают элемент относительно исходного положения.
- static — элемент устанавливается в фиксированное положение относительно фона и не будет перемещаться при прокручивании страницы.

Следующий пример показывает использование свойства z-index. Хотя в тексте программы изображение описано выше остальных элементов, за счет присвоения ей индекса с большим номером она перемещается поверх первого текста (рис. П1.22). Таким образом, использование свойства z-index освобождает от естественного порядка упоминания элементов в тексте HTML-программы.

```
^ПThL •*
<HEAO><T1T1E>Позиционирование</TITLE></HEAD>
<BODY BGCOLOR="AQUA">
<DIV STYLE="position:absolute; top:0;left:70;width:50;height:100;z-index :2">
<IMG SRC="logotip.gif">
</DIV>
<DIV STYLE="position:absolute;top:10;lett:15;width;400;height:100">
<H1 STYLE="color : гей">Первый позиционированный текст, который
накладывается на рисунок и на второй текст</H1>
</DIV>
<DIV STYLE="position: absolute; top : 60; left: 300;width: 50;height: 100">
<H1 5TYLE="color : Ыие">Второй позиционированный текст</H1>
</DIV>
</BODY>
</HTML>
```



Рис. П1.22. Пример позиционирования элементов

При позиционировании элементов может оказаться, что размеры элемента превышают размеры фрагмента (отводимой области, заданной в нашем примере тегом <DIV>). Например, текст или изображение не помещаются полностью в выделенный для них прямоугольник. На этот случай имеется свойство overflow (переполнение).

Свойство overflow может иметь три значения:

- попе (ничего) — если элемент и выйдет за пределы фрагмента (отведенного для него места), он все равно будет показан;
- clip — выступающие за границы фрагмента части элемента будут обрезаны;
- scroll — будет использована прокрутка.

В следующем примере используется свойство overflow для создания механизма прокрутки первого текста (рис. П1.23):

```
<HTML>
<HEAO><T1T1E>Позиционирование</T1T1_E></HEAO>
<BODY BGCOLOR="AQUA">
<DIV STYLE="position:absolute; top:0;lett:70;width: 50;height:100">
<IMG SRC="logotip.gif">
</DIV>
<DIV  STYLE="position:absolute;
top:10;left:15 ;width:220;height:120;overflow:scroll">
<H1 STYLE="color: гес!">Первый позиционированный текст, который
накладывается на рисунок и на второй текст</H1>
</DIV>
<DIV STYLE="position:absolute; top: 60;left:300;width:50;height:100">
<H1 STYLE="color:Ыие">Второй позиционированный текст</H1>
</DIV>
</BODY>
</HTML>
```

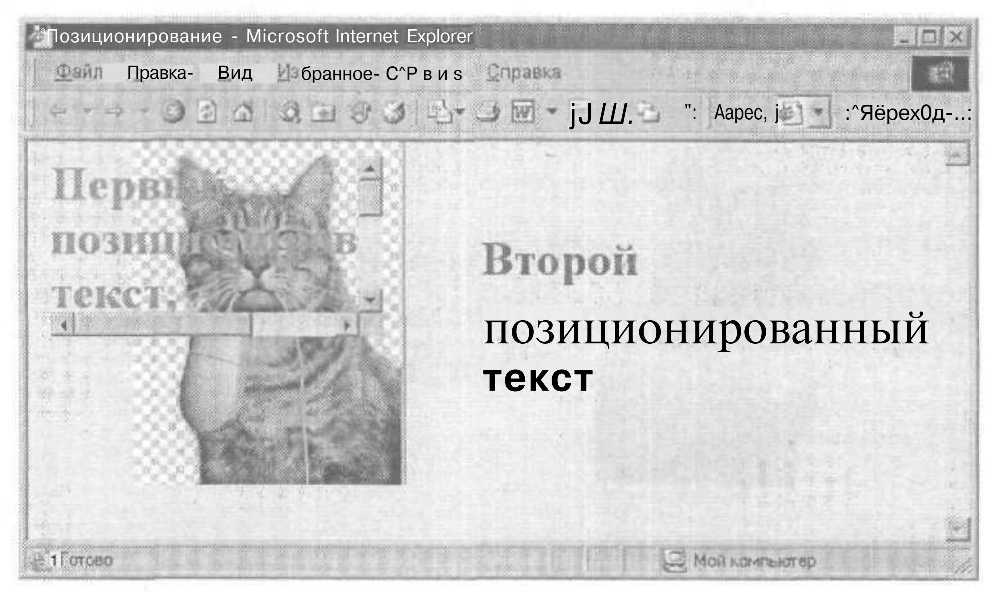


Рис. П1.23. Использование свойства overflow для создания механизма прокрутки

Понятно, что все рассмотренные выше страницы-уродцы создавались лишь с одной целью — продемонстрировать возможности языка, предоставляемые для позиционирования элементов, а не показать, что можно делать красивые вещи. Вы сами должны решить, какие средства и с какой целью использовать на своей веб-странице.

Статические фильтры

Фильтры — это эффекты изменения внешнего вида графики и текста на странице. С помощью фильтров вы можете отражать тексты и графику, создавать эффект движения, как это делается в графических редакторах. Другими словами, фильтр — это трансформация исходного изображения по определенным правилам (алгоритмам). Существуют статические и динамические фильтры. Статические фильтры изменяют внешний вид элемента, оставляя его неподвижным. Динамические фильтры позволяют трансформировать графический элемент со скоростью, задаваемой пользователем.

Сначала рассмотрим статические фильтры. Статический фильтр можно задать как свойство в таблице стилей, используя запись вида:

```
filter: название_фильтра
```

или, при наличии параметров, такую запись:

```
filter: название_фильтра(параметр1,параметр2,...,параметрN)
```

В языке HTML (версии 4) имеется 14 статических фильтров. Перечислим те из них, которые используются без параметров и чаще других:

- blur — создает эффект движения объекта;
- **fliph** — отражает объект симметрично относительно центральной горизонтальной оси;

* flipv — отражает объект симметричноотносительно центральной вертикальной оси;
* wave — выполняет синусоидальное преобразование объекта вдоль вертикальной оси;
* Xray — изменяет глубину цвета объекта и отрисовывает черно-белым, делая его похожим на рентгеновский снимок («рентгеновское изображение»).

Ниже приводится код программы, которая выводит графический файл и текст сначала без фильтра, а затем с фильтрами flipv, fliph и blur. Результат работы этой программы показан на рис. П1.24. Фильтр flipv позволяет получить вертикальное отражение изображения и текста, фильтр fliph — горизонтальное отражение, а фильтр blur — размытие. Обратите внимание, что фильтр blur, примененный к тексту, создает эффект трехмерности, то есть текст как бы отбрасывает тень.

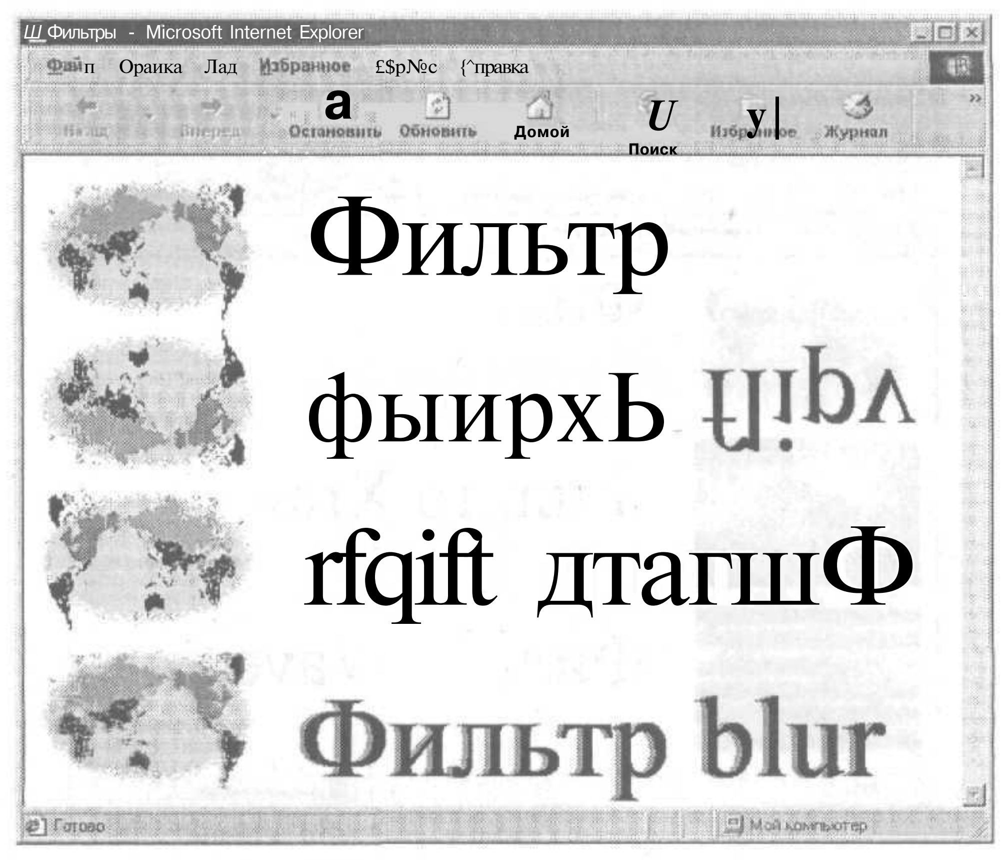


Рис. П1.24. Применение фильтров flipv, fliph и blur

```
<HTHL>
<HEAO><T1T1_E>Фильтры</TITLE></HEAO>
<STYLE>
P {position:absolute;top:0;left:190;font-size:80}
</5TYLE>
<BODY>
```

```
<IMG STYLE="position:absolute;width:150;height:100" SRC="world.gif">
<IMG
STYLE="posit ion:absolute;top:120;left:10;width:150;height:100;filter:flipv"
SRC="world.gif ">
<IMG
STYLE="posit ion:absolute;top:230;left:10;width:150;height:100;filter:fliph"
SRC="world.gif ">
<IMG
STYLE="position:absolute;top:340;left:10;width:150;height:100;filter:blur"
5RC="world.gif ">
<P>Фильтр</P>
<P STYLE="top:130;filter:flipy">Фильтр flipv</P>
<P STYLE="top:240;filter:fliph">0MnbTp fliph</P>
<P STYLE="top:360;filter:Ыиг">Фильтр blur</P>
</BODY>
</HTML>
```

Покажем работу еще двух фильтров: Xray и Wave (рис. П1.25).

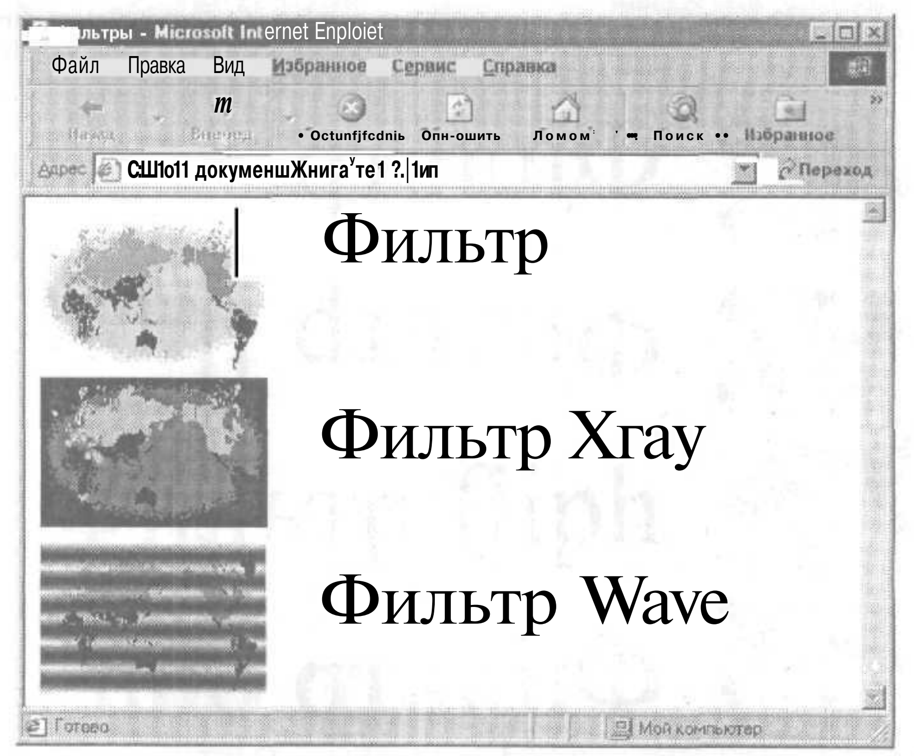


Рис. П1.25. Фильтры Хгау и wave не воздействуют на тексты

```
<HTML>
<HEAO><T1T1E>Фильтры</TITLE></HEAD>
<STYLE>
p {position:absolute;top:0;left:190; font-size:48}
</STYLE>
<BODY>
<IMG STYLE="positi on;absolute;width:150;height: 100" SRC="world.gif" >
```

```
<IMG
STYLE="position: absolute; top: 120 ; left: 10 ; w-idth: 150 ; height: 100; filter :Xray"
SRC="world.gif">
<IMG
STYLE="posit ion:absolute;top:230;left:10;width:150;height:100;filter:Wave"
SRC="world.gif ">
<P>Фильтр</P>
<P  STYLE="top:130;filter:Xray">Фипьтр Xray</P>
<P STYLE="top:240;fHter:Wave">OnnbTpWave</P>
</BODY>
</HTML>
```

Динамические фильтры

Эффекты постепенного появления (исчезновения) графического изображения и преобразования одного изображения в другое можно получить с помощью совместного применения динамического фильтра и сценария. Разумное использование таких эффектов украшает веб-страницу. Существует несколько способов преобразований. Эти способы заранее определены, поэтому от вас требуется лишь указать номер выбранного способа.

В табл. **П1.4** приведены названия и номера превращений.

Таблица П1.4. Типы воздействия фильтров и их номера

Тип превращения	Номер фильтра
Стягивающийся прямоугольник	О
Расширяющийся прямоугольник	1
Стягивающийся круг	2
Расширяющийся круг	3
Стирание вверх	4
Стирание вниз	5
Стирание вправо	б
Стирание влево	7
Вертикальные жалюзи	8
Горизонтальные жалюзи	9
Сужающиеся клетки	10
Закрывающаяся шахматная доска	11
Случайный наплыв	12
Вертикальное деление внутрь	13
Вертикальное деление наружу	14
Горизонтальное деление внутрь	15
Горизонтальное деление наружу	16
Стирание влево-вниз	17
Стирание влево-вверх	18
Стирание вправо-вниз	19

продолжение ⇨

Таблица П1.4 *(продолжение)*

Тип превращения	Номер фильтра
Стирание вправо-вверх	20
Случайные горизонтальные полосы	21
Случайные вертикальные полосы	22
Случайный выбор номера (из 0-22)	23

В следующем примере рисунок с изображением карты мира постепенно становится видимым через открывающиеся вертикальные жалюзи (рис. П1.26). Изображение из файла word.gif сначала невидимо (свойство visibility:hidden таблицы стилей). В теге <DIV> применен динамический фильтр revealtrans, управляющий появлением изображений. Сценарий содержит одну функцию dyn_filter(), которая применяет фильтр (метод applyQ), делает объект видимым (свойству visibility присваивается значение "visible"), устанавливает тип преобразования (Transition=8) и задает длительность преобразования 2 с (метод playQ, которому передан числовой параметр 2). Далее определено, что эффект постепенного появления картинки начнется сразу же после загрузки страницы. Это достигается привязкой функции dyn_filter() к событию ONLOAD в теге <BODY>.



Рис. П1.26. Динамическое преобразование картинки с помощью фильтра «вертикальные жалюзи»

```
<HTM1_>
<HEAD> <TITLE> Динамический фильтр </TITLE>
<SCRIPT >
function dyn_filter()
{
```

```
Imagel.filters(O) .apply()
Il.style.visibility="visible"
Imagel.filters(O).Transition=8
Imagel.filters(Q).play(2)
}
</SCRIPT>
</HEAD>
<BODYONLOAD="dyn_filter()">
<DIV ID=Imagel STYLE="position:absolute;
height:200;width:300;left:10;top:10;filter:revealtrans">
<IMG ID=I1 STYLE="position:absolute;
height:200;width:300;visibility:hidden" SRC=world.gif>
</DIV>
<H3 STYLE="position:absolute;top:210">
Динамическое преобразование картинки с помощью фильтра
</H3>
</BODY>
</HTML>
```

Заметим, что Imagel — это имя (идентификатор) элемента-контейнера, заданного тегом <DIV>. Контейнер содержит единственный элемент (изображение), но мы использовали его только затем, чтобы применить к нему метод applay(), который не поддерживается элементом, созданным тегом <IMG>. Filters — коллекция всех фильтров. Фильтр (единственный) указан номером в коллекции: Imagel.filters(O). Если в приведенной выше программе изменить тип преобразования, например, на 3 (расширяющийся круг) путем замены в теле функции соответствующей записи Imagel.filters(0).Transition=3,то получится эффект, показанный на рис. П1.27 (изображение постепенно появляется в расширяющемся круге).



Рис. П1.27. Динамическое преобразование изображения с помощью фильтра «Расширяющийся круг»

При установке фильтра типа 12 возникает эффект постепенного повышения разрешения изображения путем случайного заполнения пикселами места, отведенного под рисунок. Точки изображения постепенно выводятся на экран в случайном порядке.

Тип преобразования 23 — случайный выбор и применение одного из имеющихся типов (от 0 до 22). Заранее вы не сможете угадать, какой именно фильтр сработает, когда пользователь загрузит вашу страницу.

Следующая программа аналогична рассмотренной выше, но в ней применяется другой фильтр (blendtrans), который создает эффект постепенного повышения яркости и насыщенности изображения. Этот фильтр имеет только один параметр - продолжительность преобразования (duration). Обратите внимание, что новая программа получилась из предыдущей путем замены двух строк.

```
<HTML>
<HEAD> <TITLE> Динамический фильтр </TITLE>
<SCRIPT >
function dyn_fi lter()
{
Imagel.fl iters(0).Apply()
II.style.visibillty="visible"
Imagel. fiIters(0).play(2)
}
</SCRIPT>
</HEAD>
<BODY ONLOAD="dyn_filter()">
<DIV ID=Imagel STYLE="position : absolute;
height:200;width:300;left:10;top:10; fiIter:blendtrans" >
<IMG ID=I1 STYLE="position:absolute;
height: 200;width: 300;visibili ty :hidden" SRC=world. gif>
</DIV>
<H3 STYLE="position:absolute;top:210">
Динамическое преобразование картинки с помощью фильтра
</H3>
</BODY>
</HTML>
```

Попробуйте применить фильтр revealtrans, управляющий появлением изображения, при различных значениях (0-23) параметра transition, который задает тип преобразования.

Для этого можно использовать в качестве основы один из рассмотренных выше текстов HTML-программы.

При создании динамических сцен может потребоваться изменение ориентации рисунка при изменении направления его движения. Например, при изменении направления движения самолета на противоположное следует развернуть его изображение на 180°. Конечно, можно иметь два рисунка с самолетом, на одном из которых самолет летит направо, а на другом — налево. Тогда вам потребуется только динамически изменять аргумент атрибута SRC в теге <IMG>. Другой способ — поместить два рисунка на одном и том же месте, но динамически управлять их видимостью так, чтобы в любой момент отображалось только одно изображение. Наконец, можно просто применить соответствующий статический фильтр к одному рисунку. Попробуйте самостоятельно разработать эти сценарии.

Таблицы стилей в отдельных файлах

При использовании тега <STYLE> необходимо вставлять таблицу стилей в каждый документ. Это может показаться нерациональным как с точки зрения объема файлов, так и с точки зрения времени, необходимого для разработки страницы. Однако существует способ автоматического применения таблицы стилей, сохраненной в отдельном файле. Имя этого файла должно иметь расширение .ess. Для применения стилей, описанных в этом файле, к данному документу используется специальная инструкция, которую следует вставить между тегами <STYLE> и </STYLE>. Вот формат этой инструкции:

```
@import URL("адрес_файла_стилей")
```

Например:

```
@import URL("http://abcd/xyz/sty1e.css")
```

Здесь приведен вымышленный, реально не существующий адрес файла описания стилей.

Другой способ использования внешних таблиц стилей основан на применении тега <LINK> внутри тега <HEAD>. Например:

```
<HEAD>
<LINK REL=STYLESHEET TYPE="text/ess" MEP="адрес_таблицы_стилей"
TITLE="Style"
</HEAD>
```

Вставка Flash-мультфильма в веб-страницу

Flash-мультфильм содержится в SFW-файле, то есть в файле с расширением .swf, который создается в пакете Macromedia Flash. Чтобы вставить его в веб-страницу (в соответствующий ей HTML-документ), необходимо написать несколько строк HTML-кода. А именно нужно вставить объект, который будет проигрывать (показывать) Flash-файл. Flash-файл может содержать мультимедийный документ (анимацию, векторную и растровую графику, звук, элементы управления, поддерживающие интерактивность). В частности, вы можете создать статическое изображение, содержащее много элементов (например, большой чертеж). Для этого сохраните его в векторном экономном SWF-формате и вставьте в свой HTML-документ. Более того, Flash-мультфильм может почти полностью определять и представлять содержание вашей веб-страницы, оставляя HTML-документулишь роль контейнера.

В пакете Macromedia Flash имеется специальная команда для создания HTML-документа, содержащего все необходимые теги, обеспечивающие проигрывание SWF-файла, — Publish (Публикация). Однако нередко требуется вставить готовый мультфильм в уже имеющуюся веб-страницу. В этом случае удобнее скорректировать HTML-документ вручную, с помощью обыкновенного текстового редактора.

Итак, чтобы вставить Flash-мультфильм в HTML-документ, необходимо написать в этом документе несколько строк, задающих объект, который будет проигрывать (показывать) мультфильм. Это тег <OBЗECT> с соответствующими параметрами. Тег <O BЗ ECT> является контейнером, то есть тегом, который содержит другие теги,

задающиепараметры.Главныйпараметртега<OBJECT> —classid.Онуказываетна Flash-проигрыватель (элемент управления ActiveX). Параметр codebase указывает, где в сети взять Flash-проигрыватель, если он не установлен на вашем компьютере. Тег <EMBED> вставлен из-за Netscape-браузера. Другие параметры объекта записываются в теге <PARAM>. Отметим лишь основные из них, которых в большинстве случаев хватает. Параметр <PARAM NAME=movie VALUE="tfMfl_swf_c}>aiwa"> указывает на имя swf-файла с Flash-мультфильмом. Параметры WIDTH и HEIGHT (ширина и высота) определяют размеры прямоугольника, в котором будет размещаться ваш Flash-мультфильм. Имейте в виду, что мультфильм может быть обрезан, ему также может быть отведено слишком много места на странице. Параметр <PARAM NAME=wmode VALUE=transparent> определяет, каким будет фон вашего ролика. В частности, значение transparent задает прозрачность фона. Это значение наиболее часто используется при вставках мультфильмов в веб-страницы. Все возможные значения параметров лучше всего изучить по книгам, посвященным пакету Flash. Для точного позиционирования Flash-роликанасвоей странице можно воспользоваться контейнером с заданием координат в атрибуте STYLE. Например, в Internet Explorer для этого подходит контейнерный тег<DIV>.

В качестве примера приведем вставку Flash-мультфильма, представляющего собой калькулятор. Файл этого мультфильма, calculator.swf, можно взять из коллекции примеров пакета Flash 5.0. HTML-код имеет следующий вид:

```
<DIV style="position:absolute; top: 120; left: 100">
<OBJECT class1d="clsid:D27CDB6E-AE6D-llcf-96B8-444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/
swflash.cab#version=6,0,0,0"
WIDTH="430" HEIGHT="450" id="Flashl" ALIGN="">
<PARAM NAME=movie VALUE = "calculator.swf">
<PARAM NAME=quality VALUE=high>
<PARAM NAME=scale VALUE=noborder>
<PARAM NAME=wmode VALUE=transparent>
<PARAM NAME=bgcolor VALUE=#FFFFFF>
<EMBED src="Flashl.swf" quality=high scale=noborder wmode=transparent
bgcolor=#FFFFFF WIDTH="430" HEIGHT="450" NAME="Flashl" ALIGN=""
TYPE="application/x-Shockwave-flash" PLUGINSPAGE="http://
www.macromedia.com/go/getflashplayer"></EMBED>
</OBJECT>
</DIV>
```

Для изучения всех возможных параметров советуем обратиться к пакету Flash и поэкспериментировать с командой Publish (Публиковать) при различных вариантах параметров. Выбирая различные значения параметров, следите за тем, какой HTML-код получается при публикации.

Вставка звука и видео

Браузеры могут воспроизводить звуковые файлы. Например, Internet Explorer может загружать и воспроизводить фоновый звук (для его прослушивания не нужно щелкать на ссылке). Звук хранится в файле. Internet Explorer распознает форматы звуковых файлов WAV, AU и MIDI. Для задания фонового звука в Internet Explorer используется тег следующего формата:

```
<BGSOUND 811C="звуковой_файл" Ю0P=число>
```

Атрибут LOOP может принимать следующие значения:

- TRUE — бесконечное повторение, пока страница на экране;
- FALSE — воспроизведение звукового файла один раз;
- число — указание конкретного числа воспроизведений.

Пример

```
<BGSOUND 5KC="симфония.wav" L00P=5>
```

В HTML-документах можно использовать ссылки на звуковые и видеофайлы, которые будут воспроизводиться при активизации ссылок. Кроме того, можно использовать специальный тег <EMBED> для размещения панели проигрывателя (плейера) на странице сразу же при ее загрузке в браузер.

Для различных платформ существуют разнообразные форматы звуковых и видеофайлов, но в Windows общепринятым звуковым форматом является WAV, а в Macintosh — AIFF. Для видео общепринят формат MPEG. Он используется в Windows, Macintosh и Unix. Формат QuickTime также широко распространен и может использоваться на платформах Windows и Macintosh. Можно выбирать и другие форматы, однако следует предупреждать об этом пользователей, а также помнить, что звуковые и видеофайлы могут быть очень большого объема, поэтому для их загрузки может потребоваться много времени. В связи с этим желательно сообщать пользователям размеры аудио- и видеофайлов, чтобы они могли решить, стоит ли тратить время на их загрузку.

Особо отметим технологию передачи и воспроизведения звука в режиме реального времени (RealAudio). В настоящее время она получила широкое распространение в Интернете. Эта технология позволяет устраивать аудио- и видеоконференции, осуществлять вещание радиостанций в Интернете. Передача широкополосных звуковых сообщений по узкополосным телефонным линиям практически в реальном времени основана на применении различных алгоритмов сжатия. В настоящее время наиболее популярны программы RealPlayer G2 и RealPlayer Plus G2. Звуковые файлы, используемые технологией RealAudio, имеют расширение RA. Кроме того, могут использоваться метафайлы с расширением RAM. Метафайлы являются обычными текстовыми файлами, каждая строка которых содержит полный URL-адрес звукового файла с расширением RA. Первым элементом такого адреса (указывающим на тип протокола) является pnp.

Пример

```
pnp: //audio.real.c om/example.ra
```

Плейер RealPlayer G2 (или RealPlayer Plus G2) может воспроизводить не только аудио-, но и видеофайлы различных форматов.

Рассмотрим встраивание звуковых и видеофайлов более подробно.

Пример ссылки на видеоклип:

```
<A HREF="ocean.qt">Cep(t)HHr, видеоклип, 520K</A>
```

В тексте этой ссылки мы указали размер видеоклипа — 520K, чтобы пользователь оценил время его загрузки в браузер.

Internet Explorer распознает атрибут DYNSRC тега <IMG>, который позволяет встраивать видео следующим образом. На странице выводится изображение, при наведении на которое указателя мыши начинается воспроизведение видеоклипа. Вот один из основных вариантов формата:

```
<IMG ОУМ5(*С="видео_файл" START=MOUSEOVER ЮОР=чиспо_воспроизведений>
```

Пример

```
<IMG DYNSRC="clip.avi" START=MOUSEOVER LOOP=3>
```

Нижеприведен пример встраивания звукового файлаexample.wavспомощью тега <EMBED> и ссылки на этот файл. Тег <EMBED> позволяет сразу разместить панель проигрывателя звуковых файлов, а в случае использования ссылки панель проигрывателя появляется на экране только при щелчке на этой ссылке.

```
<HTM1_>
<HEAD><TITLE>ripnMep использования :-iByKa</TITLE></HEAD>
<BODY>
        <B>Встраивание звукового файла</B>
<P><EMBED SRC="example.wav" >
<P>
<P><EMBED SRC="example.wav" HEIGHT=150 WIDTH=180>
<P><A HREF="example .wav"B>CcbmKa на звуковой файл</A>
</BODY>
</HTML>
```

Поле ввода данных

При создании интерактивных страниц может потребоваться передать ряд символов от пользователя. Например, мы можем попросить пользователя ввести его адрес электронной почты. Для этого необходимо поле ввода. Затем все то, что было введено, можно обработать с помощью программы-сценария. Чтобы организовать поля вводаданных, применяется тег<INPUT>с некоторыми атрибутами (рис.П1.28).

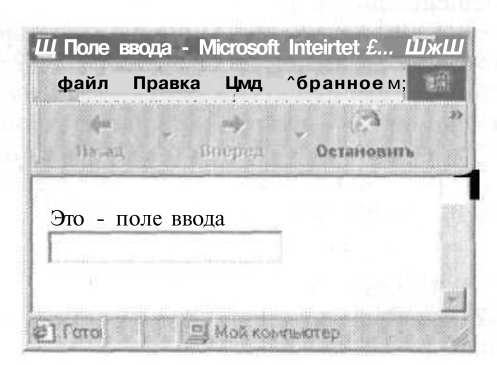


Рис. П1.28. Поле ввода

Для ввода строки символов формат тега <INPUT> имеет вид:

```
<INPUT TYPE="TEXT">
```

Если вы хотите, чтобы введенные символы появлялись на экране в виде звездочек (что обычно нужно при вводе пароля), то используйте следующий формат:

```
<INPUT TYPE="PASSWORD">
```

Для ввода числовых значений используется соответствующий аргумент атрибута TYPE:

```
<INPUT TYPE="NUMERIC">
```

В тег<INPUT> можно вставить и другие атрибуты:

- MAME="имя" — имя переменной, в которой сохраняется введенное значение;
- \/АШЕ="значение" — начальное значение;
- 512Е="число" — длина текстового поля;
- MAXLENGTH="4Mcno" — максимальное количество символов, которое можно ввести.

Например:

```
<INPUT TYPE="TEXT" NAME="USERTEXT" VALUE="" SIZE="20">
```

Существует и другое средство для предоставления пользователю возможности вводить данные — стандартная функция promptQ языка JavaScript, которая принимает в качестве параметров пояснительный текст и начальное значение, а затем отображает на экране окно для ввода значения. В этом окне есть две кнопки — OK и Отмена. Функция возвращает введенное пользователем значение либо false, если пользователь нажал кнопку Отмена.

Например, в результате выполнения функции prompt("Введите текст","") появляется окно, представленное на рис. П1.29.

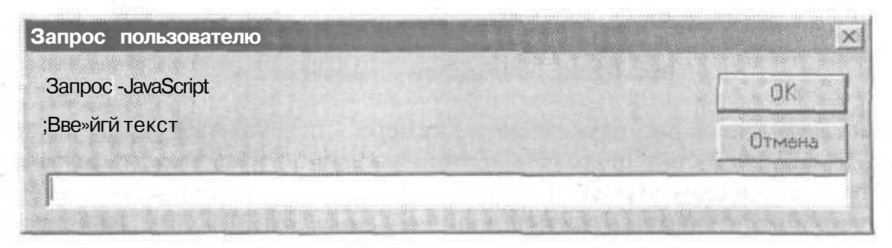


Рис. П1.29. Название рисунка

Переключатели (radiobuttons)

Тег <INPUT> позволяет вывести на страницу не только поле ввода, но и другие элементы. Например, если использовать атрибут TYPE="RADIO", то можно создать набор переключателей (radiobuttons) — это кружки, щелчок на одном из которых делает последний отмеченным, а все остальные — неотмеченными. Чтобы набор переключателей воспринимался как целое, его необходимо поместить внутри тега списка <UL> и в каждом теге <INPUT> использовать одно и то же имя (значение атрибута NAME). Чтобы выделить переключатель, применяется атрибут CHECKED. После тега <INPUT> помещается текст, который мы хотим видеть рядом с переключателем. Атрибут VALUE используется для того, чтобы мы смогли узнать, какой переключатель выбрал пользователь, и обработать выбор в программе-сценарии.

В следующем примере создается набор из трех переключателей (рис. П1.30):

```
<HTML>
<HEAO><T1T1E>Переключатели</T1T1E></HEAO>
<H2>Какой язык вы используете?
<UL>
<INPUT TYPE= "RADIO" NAME="LANG" УА1_иЕ = "PyccKHvi" CHECKED> Русский<BK>
```

```
<INPUT TYPE= "RADIO" NAME="LANG" VALUE="Английский"> Английский <BR>
<INPUT TYPE="RADIO" NAME="LANG" VALUE = "Немецкий" > Немецкий
</UL>
</H2>
</HTML
```

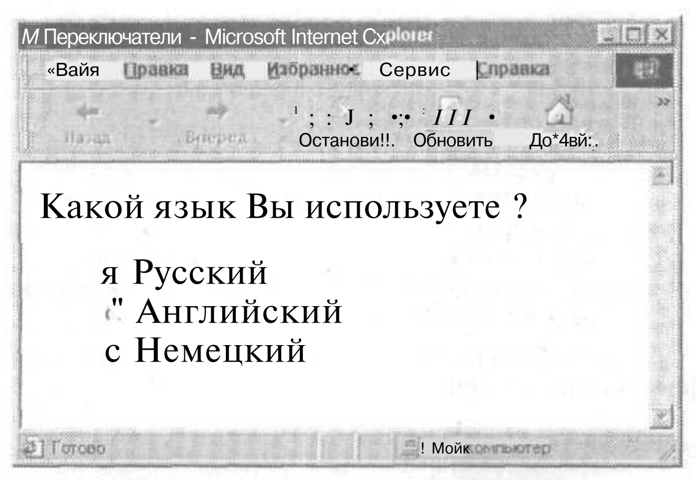


Рис. П1.30. Использование переключателей

Элементы списка (в рассматриваемом примере — переключатели) располагаются по умолчанию в столбце, то есть вертикально. А как расположить их горизонтально, то есть в одну строку? Горизонтальное расположение элементов списка довольно часто встречается на страницах, поскольку это позволяет экономить место. Один из возможных и часто используемых способов — применение тега таблицы как метода позиционирования. Ниже приводится вариант соответствующей программы и результат ее работы (рис. П1.31).

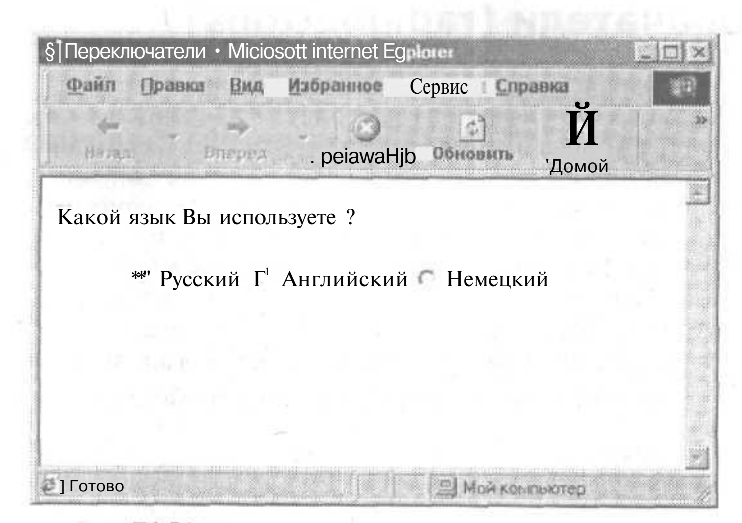


Рис. П1.31. Расположение переключателей в одну строку

```
<HTML>
<HEAO><T1T1E>Переключатели</T1T1E></HEAO>
Какой язык вы используете?
<UL>
<TABLE>
<TR>
<TD>
<INPUT TYPE= "RADIO" NAME="LANG" VALUE="PyccKHU" CHECKEO>Русский
</TD>
<TD>
<INPUT TYPE="RADIO" NAME="LANG" УА1.иЕ="Английский"> Английский
</TD>
<TD>
<INPUT TYPE="RADIO" NAME="LANG" VALUE="Немецкий" > Немецкий
</TD>
</TR>
</TABLE>
</UL>
</HTML>
```

Флажки

Флажки отличаются от переключателей тем, что в одном наборе можно установить (отметить) более одного флажка (рис. П1.32). Задаются флажки так же, как и переключатели, однако аргументом атрибута TYPE должен быть аргумент CHECKBOX:

```
<HTML>
<HEAD><TITLE> Флажки< /T1T1_E></HEAO>
<H2>Какие экзамены вы будете сдавать?
<UL >
<INPUT TYPE = "CHECKBOX" NAME= "TEST" VALUE= " Физика" CHECKED> Русский<ВИ>
<INPUT TYPE="CHECKBOX" NAME="TEST" УАИ)Е= "Английский язык"> Английский
язык<BR>
<INPUT TYPE="CHECKBOX" NAHE="TEST" VALUE="Математика" > Математика
</UL>
</H2>
</HTML>
```

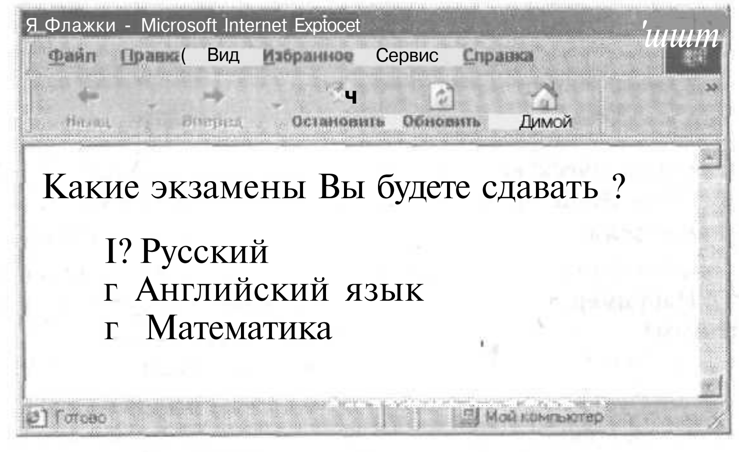


Рис. П1.32. Пример использования флажков

Если флажки необходимо расположить горизонтально, используйте тег <TABLE>, как в предыдущем примере.

Кнопки

Настранице можно разместить кнопки, щелчок клавишей мыши на которых обрабатывается программой-сценарием. Например, вы можете создать кнопку с надписью Поиск. Что произойдет, если пользователь нажмет эту кнопку, зависит от вашего сценария. Кнопка создается с помощью тега **<BUTTON>**. Мы можем поместить на кнопку текст и изображение, а также позиционировать ее в нужное место с помощью атрибута STYLE (рис. П1.33).

В следующем примере создается кнопка с изображением из файла **logotip.gif** и надписью:

```
<HTML>
<BUTTON STYLE="posi tion : absolute ; wi dth : 150 ; hei ght: 60">
<IMG SRC="logotip.gif" WIDTH="50" HEIGHT="50">
Нажми меня
</BUTTON>
</HTML>
```

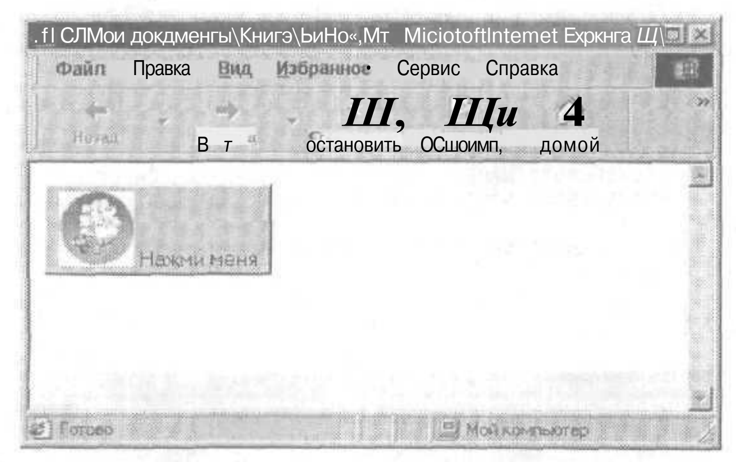


Рис. П1.33. Кнопка с изображением и надписью

Можно создать графическую кнопку, используя уже знакомый тег <INPUT>, но с атрибутами **TYPE="IMAGE"** и **5КС="имя_файла",** а также NAME и VALUE:

```
«INPUT TYPE="IMAGE" 51?C="имя_файла" MAME="имя_кнопки" УА1иЕ=значение>
```

В качестве рисунка можно подобрать изображение кнопки или любое другое (рис. **П1.34).** Например, следующая строка кода выводит стрелку из файла значка (пиктограммы):

```
<INPUT 1YPE="IMAGE" SRC="arwl0ne.ico" NAME="N" VALUE=значение>
```

Кнопку также можно создать с помощью тега:

```
<INPUT TYPE="BUTTON">
```

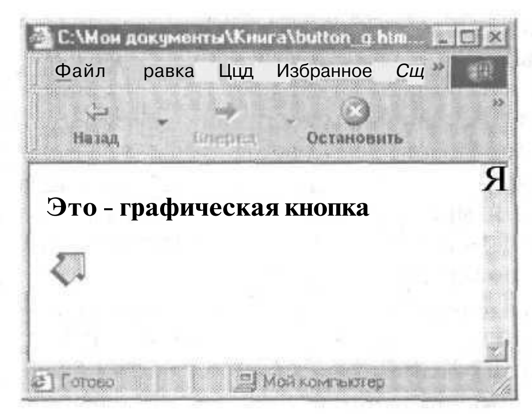


РИС. П1.34. Пример графической кнопки

Кроме рассмотренных выше, можно задать специальные кнопки для отправки данных серверному приложению (CGI) и очистки полей ввода (и восстановления значений, принятых по умолчанию). В HTML-документ можно включить несколько элементов типа полей ввода данных, переключателей и флажков. Если они входят в один смысловой блок, то эти элементы следует заключить в тег формы **<FORM>**. В этот же тег можно включить и специальные кнопки.

Если мы хотим отправлять данные серверному приложению, то тег <FORM> должен содержать атрибут, указывающий способ передачи, и атрибут, содержащий адрес серверного приложения.

Пример

```
<FORM METHOD="POST" ACTION="/bin/serv">
```

Внутри этого тега можно определить кнопку отправки.

Пример

```
<INPUT TYPE=SUBMIT VALUE="OTnpaBHTb">
```

Здесь аргументом атрибута **VALUE** является текст надписи на кнопке.

Кнопка очистки задается так:

```
<INPUT TYPE=SUBMIT RESET="04HCTHTb">
```

Мы не рассматриваем подробно работу с серверными приложениями, поскольку это отдельная тема. Однако неплохо знать, что такая возможность имеется.

Фреймы

Часто возникает необходимость разместить в окне браузера несколько разделов или окон, называемых фреймами. В каждом фрейме отображается свой HTML-документ. Например, в одном фрейме можно поместить ссылки на документы, которые должны показываться в другом фрейме. Иначе говоря, в одной части окна имеется оглавление большого документа, а в другой — содержимое документа, ссылка на который была использована. Фреймы могут работать независимо, а также содержать ссылки друг на друга.

При использовании фреймов потребуется несколько HTML-файлов. Один из них называется установочным. В нем описывается расположение (раскладка) фреймов в окне браузера и назначаются исходные HTML-файлы для каждого из фреймов, но отсутствуют собственно текст и графика. Исходные HTML-файлы выводятся во фреймы и могут содержать тексты, графику, ссылки и пр.

Установочный HTML-файл, как и любой другой, начинается с тега <HTML> и заканчивается тегом </HTML>. Для разделения окна на несколько фреймов используются теги <FRAMESET> и </FRAMESET>. Тег <FRAMESET> должен быть размещен после тега </HEAD>, но перед тегом <BODY>.

Два фрейма можно расположить рядом друг с другом или друг над другом. Для задания способа расположения используется атрибут COLS (если рядом) или ROWS (если друг над другом). Чтобы разделить окно на два фрейма, указывают через запятые два числа. Эти числа определяют размеры фреймов. Для трех фреймов задаются три числа. Если нужно указать, что фрейм занимает все оставшееся место, используется символ «звездочка» (*). Размеры фреймов измеряются в пикселах или процентах. В последнем случае около числа пишется символ процента (%).

Например, тег<FRAMESET COLS="50,* "> задает деление окна на два вертикальных фрейма, первый из которых имеет ширину 50 пикселов, а второй занимает все оставшееся место.

Тег <FRAMESET ROWS="20%,30%,*"> задает разбиение на три горизонтальных фрейма высотой 20%, 30% и 50%.

Можно использовать одновременно и горизонтальное, и вертикальное разбиение окна на фреймы. Это делается вложением тегов <FRAMESET> друг в друга, то есть размещением фреймов внутри фреймов.

Задав расположение фреймов, следует указать для каждого из них исходный HTML-файл. Напомним, что исходные файлы выводятся во фреймах. Для этой цели служит тег <FRAME>, имеющий множество атрибутов, которые управляют свойствами фреймов. Ниже перечислены эти атрибуты:

- SRC="MMfl_4)aMa" MAME="имя_фрейма" — каждый фрейм должен иметь имя, упомянутое в атрибуте NAME, и к нему должен быть привязан HTML-файл, указанный в атрибуте SRC.
- SCROLLING — определяет, будут ли присутствовать в окне фрейма полосы прокрутки; если требуется прокрутка, то задается аргумент YES, иначе — N0.
- NORESIZE — запрещает пользователю изменять размеры фреймов; если этот атрибут не применяется, то пользователь не может изменять размеры.
- BORDER="uiMpMHa_ncwocbi" — определяет ширину разделительной полосы между фреймами в пикселах.
- BORDERCOLOR="uBeT_n(„ocbi • - определяет цвет разделительной полосы; цвет задается либо шестнадцатеричным кодом, либо именем.
- MARGINHEIGHT="BbicoTa_BepXHero_OTcryna" — добавляет пустое поле между верхней границей фрейма и началом текста или графики; измеряется в пикселах.
- МАКСИ/10TH="ширина_боковых_отступов" — добавляет пустое поле между боковыми границами фреймов и началом текста или графики; измеряется в пикселах.

Поскольку фреймы поддерживаются не всеми браузерами (Internet Explorer поддерживает), постольку необходимо использовать тег <NOFRAME> перед тегом тела

<BODY>, а в теле программы, то есть между тегами <BODY> и </BODY>, можно вставить сообщение, которое будет появляться в окне, если браузер не поддерживает фреймы. Ниже мы еще рассмотрим применение тега <NOFRAME>.

Пусть, например, требуется разделить окно браузера на два вертикальных фрейма так, чтобы в первом из них появлялось содержимое файла первый.htm, а во втором фрейме — содержимое файла второй.htm. Назначим фреймам имена соответственно "ЛЕВЫЙ" и "ПРАВЫЙ". Разумеется, имена выбираются произвольно. Ширину первого фрейма определим в 40%, а для второго выделим все оставшееся место окна. В первом фрейме запретим появление полосы прокрутки. Ширину разделительной полосы зададим равной 10 пикселам. Тогда установочный HTML-файл будет содержать следующие теги (листинг П1.6).

Листинг П1.6. Код установочного файла для создания фреймов

```
<HTML>
<HEAO><T1T1E>Пример установочного файла</T1T1E><HEAO>
< FRAME SETCOLS= "40%, *">
<FRAME SRC="nepBbiH.htm" MME="ЛЕВЫЙ" SCROUING=NO BORDER=10>
<FRAME SRC="BTOpou.htm" MAVE ="ПРАВЫЙ">
</FRAMESET>
<NOFRAME>
<BODY>
Для просмотра необходим браузер, поддерживающий фреймы
</BODY>
</NOFRAME>
</HTML>
```

Теперь надо создать два исходных HTML-файла. Пусть, например, они будут такими:

```
<HTML>
<HEAD><TITLE>nepBbm</TITLE><HEAD>
<BODY>
<H2>
Это левый фрейм, который имеет ширину 40%
</H2>
</BODY>

</HTML>

<HTML>
<HEAD><TITLE>BTOpo&</TITLE><HEAD>
<BODY>
<H2>
Это правый фрейм, который занимает оставшуюся часть окна
</H2>
</BODY>
</HTML>
```

Если загрузить в браузер установочный HTML-файл, то он будет выглядеть, как на рис. П1.35.

Во фреймах можно располагать ссылки, указывающие на фреймы. Щелчок на таких ссылках может изменить файл, выводимый в другом фрейме. Этот прием часто используется при создании так называемых навигационных панелей. Навигационная панель представляет собой фрейм, расположенный обычно вверху или по краю окна. В ней содержатся ссылки, при использовании которых во втором фрейме, большем по размеру, выводятся различные документы. Именно для этого и был предусмотрен атрибут NAME в теге <FRAME>.

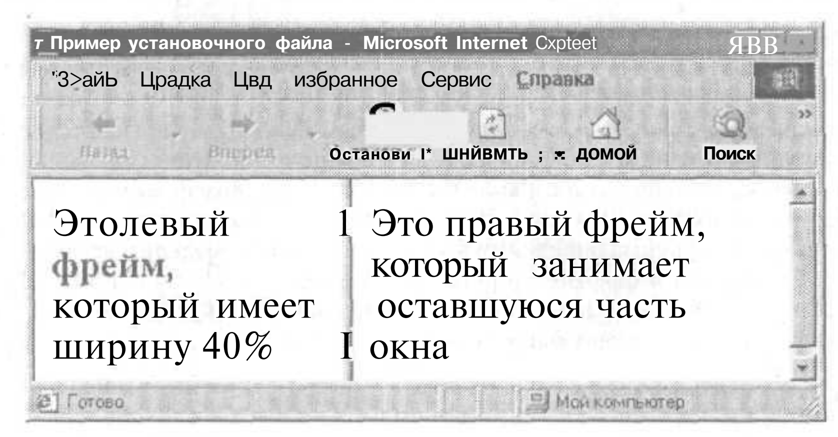


Рис. П1.35. Окно браузера разделено на два вертикальных фрейма

Ссылки на фреймы имеют формат:

```
<A HREF = "адрес_файла" TARGET="имя_фрейма"> текст_ссылки</A>
```

для текстовых ссылок **или:**

```
<A HREF="адрес_файла" TARGET="имя_фрейма"> <IMG SRC="графический_файл">
</A>
```

для графических ссылок.

Например:

```
<A HREF="new.htm" TARGET="riPABOE"> Что нового</A>
```

Здесь содержится ссылка на файл new. htm, который следует показывать во фрейме с именем ПРАВОЕ. При этом на месте ссылки появляется текст Что нового (target с английского переводится как «цель»).

При разработке многооконных страниц следует внимательно следить затем, чтобы ссылки на фреймы использовали только имена фреймов, определенные в установочном файле в теге:

```
<FRAME SRC=... NAME=...>
```

Мы рассмотрели способ создания многооконных веб-страниц. Такие страницы основаны на фреймах и состоят из установочного HTML-файла и исходных HTML-файлов.

Кроме этого, страница может включать текстовые, графические, звуковые и видеофайлы. В качестве упражнения создайте трехфреймовую страницу, подобную рассмотренной выше. Один фрейм будет содержать ваше имя, а также, возможно, фото и адрес, другой фрейм — навигационную панель со ссылками, а третий - отображать документы, на которые указывают ссылки.

Проектирование страницы может идти «сверху вниз» или «снизу вверх». Метод «сверху вниз» заключается в создании сначала установочного файла и навигационной панели. После отладки этой части программы создаются конечные файлы документов, такие как тексты, графика и т. п. Метод «снизу вверх» предполагает создание в первую очередь конечных файлов, а затем программы, организующей их вывод на экран. Выбор метода зависит от вкуса.

Тег <META>

Дополните свою веб-страницу информацией, которая заключается в тегах<META>. Эта информация не отображается браузером, однако имеет большое значение. В частности, она позволяет задать кодовую страницу языка просмотра документа, параметры его кэширования, ключевые слова, по которым вашу страницу будут искать поисковые системы, и т. д.

Метаданные размещаются на странице между тегами <HEAD> и </HEAD>.

Общий формат метатегов в HTML-документе:

```
<HTML>
<HEAD>
<TITLE>...<TITLE>
<! HTTP-эквиваленты >
<META HTTP-EQUIV=" ..." CONTENT=". . .">
<! другие теги группы HTTP-EQUIV >
...
<! группа NAME >
<META NAME="..." CONTENT="...">
<! другие теги группы NAME >
...
</HEAD>
<BODY>
...
</BODY>
</HTML>
```

Ниже приводится описание метатегов по группам.

Группа HTTP-EQUIV (HTTP-эквиваленты)

- EXPIRES — дата устаревания документа.

  После истечения указанного срока документ будет каждый раз загружаться заново, а не браться из кэша на вашем локальном диске. Формат даты спецификации RFC850.

  Пример

  ```
  <META HTTP-EQUIV="EXPIRES" CONTENT="Wed, 14 Feb 2001 08:21:53 GMT">
  ```

- PRAGMA — управление кэшированием.

  Возможно одно значение NO-CACHE, то есть данный документ не кэшируется браузером. В этом случае запрашиваемая страница будет браться с сервера, а не из кэша (буфера) пользовательского компьютера.

  Пример

  ```
  <META HTTP-EQUIV="PRAGMA" CONTENT="NO-CACHE">
  ```

- CONTENT-TYPE — тип документа и его кодировка.

  Выбор кодовой страницы для правильного отображения символов браузером.

  Пример

  ```
  <META HTTP-EQUIV="CONTENT-TYPE" CONTENT="text/html; charset=windows-1251">
  ```

- CONTENT-LANGUAGE — указание языка документа.

Значение этого параметра может использоваться как поисковыми роботами, так и веб-серверами.

Формат: <Язык>-<Диалект>

Примеры

```
<META HTTP-EQUIV="CONTENT-LANGUAGE" CONTENT="en-GB">
<META HTTP-EQUIV="CONTENT-LANGUAGE" CONTENT="ru">
```

- REFRESH — время (в секундах), через которое произойдет автоматическая перезагрузка документа или переход на другой документ с заданным URL.

  Формат: <время> или <время>; <URL>

  Пример

```
<META HTTP-EQUIV="REFRESH" CONTENT="5; http://www.microsoft.com">
```

- CACHE-CONTROL — управление кэшированием.

  Возможные варианты: кэширование в общем (PUBLIC) или частном (PRIVATE) кэше. Документ вообще не кэшируется (NO-CACHE) или кэшируется, но не сохраняется (NO-STORE).

  Пример

```
<META HTTP-EQUIV="CACHE-CONTROL" CONTENT="NO-STORE">
```

Группа NAME (имя)

- DESCRIPTION — описание документа. Один из наиболее важных параметров. Информация, содержащаяся в нем, влияет на результаты поиска, осуществляемого поисковыми системами. В общем случае вид результатов поиска, как правило, выглядит так:
  - URL документа.
  - J Название документа (содержимое <TITLE>.. .</TITLE>).
  - Описание документа, то есть DESCRIPTION, или начальный фрагмент HTML-документа, если DESCRIPTION отсутствует. В первом случае пользователь получает достаточно краткое и информативное описание документа, а во втором случае это может быть бессмысленный набор слов или несколько первых фраз из вашего документа.
  - G Рейтинг (коэффициент соответствия документа запросу пользователя).

  Пример

```
<META NAME="DESCRIPTION" COИTEMT="Описание данного документа, до 100
символов">
```

- KEYWORDS — ключевые слова. Набор слов и фраз, наиболее полно характеризующих данный документ, которые являются основным критерием поиска вашей страницы поисковыми системами. В конечном счете эти слова учитываются при выдаче результатов поиска и способствуют повышению рейтинга вашего сайта.

  Пример

```
<META NAME = "KEYWORDS" COMTEMT="Ключевые слова, разделенные запятой, до
1000 символов" >
```

- DOCUMENT-STATE — статус документа. Данный параметр управляет частотой индексации вашей страницы поисковыми серверами и может принимать два значения:
  - STATIC (документ статичен, то есть не меняется, и, следовательно, индексировать его нужно только один раз).

    Пример

    ```
    <META NAME="DOCUMENT-STATE" CONTENT="STATIC">
    ```
  - DYNAMIC (для часто изменяющихся документов, которые нужно реиндексировать).

    Пример

    ```
    <META NAME="DOCUMENT-STATE" CONTENT="DYNAMIC">
    ```
- ROBOTS — управление процессом индексации.

  Возможные варианты:
  - INDEX — возможность индексирования данного документа (иначе — NOINDEX);
  - FOLLOW — возможность индексирования всех документов, на которые есть ссылки в данном HTML-файле (иначе NOFOLLOW);
  - ALL — одновременное выполнение условий INDEX и FOLLOW;
  - NONE — одновременное выполнение условий NOINDEX и NOFOLLOW.

    Пример

    ```
    <META NAME="ROBOTS" CONTENT="INDEX, NOFOLLOW">
    ```
- RESOURCE-TYPE — тип ресурса. Для обычных HTML-документов значение этого параметра устанавливается равным "DOCUMENT".

  Пример

  ```
  <META NAME="RESOURCE-TYPE" CONTENT="DOCUMENT">
  ```
- UPDATED — дата обновления страницы.

  Пример

  ```
  <META NAME="UPDATED" CONTENT="25.11.01">
  ```
- URL —расположение главной страницы. Базовый URL определяет, какой документ следует индексировать (чтобы не обрабатывать «зеркала»).

  ```
  <META NAME="KEYWORDS" LANG="ru" CONTENT="meta,tag">
  ```
- AUTHOR — информация об авторе данного документа,
- COPYRIGHT — информация об авторских правах.
- GENERATOR — программа, создавшая HTML-код.

Допустимо, но не обязательно добавлять в метатеги атрибут LANG, указывающий язык данных.

Пример

```
<META NAME="KEYWORDS" LANG="ru" CONTENT="meta,tag">
```

Приложение 2. Справочник по HTML

Теги HTML

Здесь приведены наиболее часто употребляемые теги HTML, упорядоченные по категориям.

Структура документа

Тег	Описание
<!>	Определяет комментарий, который игнорируется анализатором HTML
<!DOCTYPE>	Объявляет тип и формат содержимого документа
<BASE>	Устанавливает URL-адрес исходного документа
<BODY>	Определяет начало основной части страницы
<COMMEMT>	Описывает комментарий, который не отображается
<DIV>	Осуществляет логическое разделение документа, создавая фрагмент внутри него
<HEAO>	Заключает в себе теги, содержащие невидимую информацию о документе
<HTML>	Идентифицирует документ как содержащий элементы HTML
<LINK>	Служит для обозначения связей между документами
<META>	Предоставляет различные типы невидимой информации и инструкций для браузера
<NEXTID>	Определяет параметр в теге <HEAD> для использования текстовыми редакторами
<SPAN>	Используется с таблицей стилей для определения нестандартных атрибутов текста на странице
<STYLE>	Заключает в себе таблицу стилей

Заголовки и названия

Тег	Описание
<H1>	Заголовок 1-го уровня
<H2>	Заголовок 2-го уровня

Тег	Описание
<H3>	Заголовок 3-го уровня
<H4>	Заголовок 4-го уровня
<H5>	Заголовок 5-го уровня
<H6>	Заголовок 6-го уровня
<TITLE>	Название документа, показываемое в заголовке окна браузера

Абзацы и строки

Тег	Описание
 	Вставляет конец строки
<CENTER>	Выравнивает заключенные в него компоненты по центру
<HR>	Помещает на страницу горизонтальную перекладину
<NOBR>	Запрещает обтекание текстом
<P>	Обозначает абзац
<WBR>	Вставляет мягкий перенос строки в блок текста в теге <NOBR>

Стили

Тег	Описание
<ADDRESS>	Определяет информацию об адресе, контрольной сумме и авторстве
<B>	Делает начертание текста полужирным, где это возможно
<BASEFONT>	Устанавливает шрифт, используемый по умолчанию
<BIG>	Задает размер шрифта на 1 пункт больше текущего
<BLOCKQUOTE>	Обозначает отступ в тексте
<CITE>	Выделяет текст курсивом; часто используется как знак авторского права
<CODE>	Вставляет код в виде текста с моноширинным шрифтом
<DFM>	Используется для пометки терминов, используемых в первый раз
<EM>	Выделяет текст, обычно курсивом
<FONT>	Определяет вид, размер и цвет шрифта для текста
<I>	Делает начертание текста курсивным, где это возможно
<KBD>	Используется для отображения на экране текста, который должен ввести пользователь
<LISTING>	Возвращает текст с моноширинным шрифтом
<PLAINTEXT>	Представляет текст моноширинным шрифтом без обработки тегов
<PRE>	Представляет текст моноширинным шрифтом
<S>	Выводит текст зачеркнутым
<SAMP>	Показывает текст как пример программы

Продолжение

Тег	**Описание**
<SMALL>	Определяет, что текст должен быть показан уменьшенным шрифтом относительно основного
<STRIKE>	Выводит текст зачеркнутым
<STRONG>	Выделяет текст жирным шрифтом
<STYLE>	Заключает в себе таблицу стилей
<SUB>	Выделяет текст, делая его нижним индексом
<SUP>	Выделяет текст, делая его верхним индексом
<TT>	Задает моноширинный шрифт
<U>	Делает текст подчеркнутым
<VAR>	Задает мелкий шрифт, иногда курсив, фиксированной ширины, используемый обычно для обозначения переменных
<XMP>	Показывает текст моноширинным шрифтом

Списки

Тег	**Описание**
<DD>	Используется внутри списка определений для задания текста в теге <DT>
<DIR>	Возвращает текст как список каталога
<DL>	Используется для создания списка элементов со связанными описаниями
<DT>	Обозначает определение, описание которого находится в следующем за ним теге <00>
<LI>	Обозначает один элемент в нумерованном или маркированном списке <MENU>
<OL>	Преобразует строки текста с тегами <LI> в нумерованный список
<UL>	Преобразует строки текста с тегами - 1> в маркированный список

Таблицы

Теги	**Описание**
<CAPTION>	Определяет заголовок таблицы
<COL>	Определяет вид столбцов таблицы по умолчанию
<COLGROUP>	Определяет контейнер для группы столбцов
<TABLE>	Определяет раздел тегов <TR>, <TD> и <TH>, организованных по строкам и столбцам '
<TBODY>	Определяет раздел, содержащий теги <TR> и <TD>, который формирует основную часть таблицы
<TD>	Определяет ячейки таблицы
<TFOOT>	Определяет набор строк для использования в нижней части таблицы
<TH>	Определяет строку заглавия таблицы; содержимое выровнено по центру и выводится жирным шрифтом

Теги	Описание
<THEAD>	Определяет набор строк, используемых как заголовок таблицы
<TR>	Определяет строку таблицы

Ссылки

Тег	Описание
<A>	Определяет гиперссылку; обязательно задавать атрибут HREF или NAME
<A HREF="url">	Определяет гиперссылку на элемент другого документа
<A> MAME="имя">	Определяет гиперссылку на элемент этого же документа

Графика, объекты, мультимедиа и сценарий

Тег	Описание
<APP1_ET>	Помещает апплет Java или другой выполняемый элемент на страницу
<AREA>	Определяет форму активного участка изображения-карты клиента
<BGSOUND>	Определяет фоновое звуковое сопровождение страницы
<EMBED>	Внедряет документы любого типа на страницу для просмотра соответствующим приложением
<IMG>	Используется для вставки графического элемента или видеоклипа на страницу
<MAP>	Задает коллекцию активных участков изображения-карты клиента
<MARQUEE>	Создает на странице бегущую строку текста
<NOSCRIPT>	Определяет, что будет показывать браузер, не поддерживающий сценарии
<OBJECT>	Встраивает на страницу объект или другой элемент управления, не встроенный в HTML
<PARAM>	Используется в теге <OBJECT> для установки свойств объекта
<SCRIPT>	Определяет раздел страницы, который должен обрабатываться интерпретатором кода сценария

Формы

Тег	Описание
<BUTTON>	Создает на странице кнопку HTML
<FIELDSET>	Рисует прямоугольник вокруг содержащихся в нем элементов для обозначения связанных объектов
<FORM>	Описывает форму на странице, которая может содержать другие компоненты и элементы управления
<INPUT>	Задает параметры элемента управления формы
<LABEL>	Определяет текст метки или заголовка для блока элементов управления

продолжение

Тег	Описание
<LEGEND>	Определяет текст, помещаемый в прямоугольник, созданный тегом <FIELDSET>
<OPTION>	Обозначает одну из альтернатив в элементе SELECT
<SELECT>	Определяет поле списка или раскрывающийся список
<TEXTAREA>	Обозначает поле ввода, состоящее из нескольких строк

Фреймы

Тег	Описание
<FRAME>	Определяет конкретный фрейм внутри набора фреймов
<FRAMESET>	Определяет набор фреймов, который содержит фреймы или подчиненные наборы
<IFRAME>	Используется для создания встроенных «плавающих» фреймов на странице
<NOFRAMES>	Определяет, что именно будет показывать браузер, не поддерживающий фреймы

Таблицы стилей

Единицы измерения

Есть две основные категории единиц: относительные и абсолютные (плюс процентные). Лучше использовать относительные единицы, поскольку определение абсолютных требует знания конкретной системы отображения: на какой принтер, монитор или другое устройство будет выводиться информация.

Относительные единицы

Значения em, en и ex соответствуют типографским терминам и определяют соотношение с размерами других символов; px обозначает пикселы — элементы экрана, размер которых зависит от установок монитора и видеокарты пользователя.

В Internet Explorer начиная с версии 4 em и ex заменены на pt, en — на px.

Абсолютные единицы

Это in, cm, mm, pt, pc: in дает значение в дюймах, cm — в сантиметрах, mm — в миллиметрах, pt — в пунктах (72 пункта на дюйм) и pc — в пиках (1 пика равна 12 пт). Эти единицы стоит использовать только в том случае, если вы знаете, каким будет размер рабочей области устройства вывода, так как браузеры будут пытаться показать все в натуральную величину.

Проценты

Эти цифровые значения задаются как числа (с десятичной точкой или без нее), показывающие отношение к единице длины (обычно размер шрифта текущего элемента).

Свойства динамического HTML

Свойства шрифта

Свойство font

Свойство	font
Значения	<font-size>, [/<line-height>], <font-family>
Значение по умолчанию	Не определено
Поддерживается	Всеми элементами
Работает ли механизм наследования	Да
Возможна ли процентная запись	Только для <font-size>, <line-height>

Это свойство позволяет устанавливать сразу несколько свойств шрифта в одном месте с начальными значениями, определенными для используемых свойств (то есть значение, определенное по умолчанию для <font-size>, отличается от значения по умолчанию <font-family>). Данное свойство может быть использовано с соответствующими значениями, разделенными пробелами или запятыми, если устанавливается несколько шрифтов.

Свойство font-family

Свойство	font-family
Значение	Название семейства шрифта (например, **Arial)**
Значение по умолчанию	Устанавливается браузером
Поддерживается	Всеми элементами
Работает ли механизм наследования	Да
Возможна ли процентная запись	Нет

Вы можете установить несколько возможных значений в порядке предпочтения (на тот случай, когда в браузере нет нужного шрифта). Для этого просто разделите их запятыми. Следует закончить родовым названием шрифта (допустимые значения serif, sans-serif, cursive, fantasy или monospace). Если имя шрифта состоит из нескольких слов, необходимо заключить эти слова в кавычки.

Свойство font-size

Свойство	fontSize
Значения	<absoLute>, <relative>, <length>, <percentage>
Значение по умолчанию	medium
Поддерживается	Всеми элементами
Работает ли механизм наследования	Да
Возможна ли процентная запись	Да (относительно родительского размера шрифта)

Значения для этого свойства можно задать различными способами:

- абсолютный размер: допустимые значенияxx-small, x-small, small, medium, large, x-large, xx-large;
- относительный размер: допустимые значения larger, smaller;
- длина в любых единицах измерения (см. выше);

- процентное отношение: значения представлены в процентах от родительского размера шрифта.

Свойство font-style

Свойство	font-style
Значения	normal, italic, oblique
Значение по умолчанию	normal
Поддерживается	Всеми элементами
Работает ли механизм наследования	Да
Возможна ли процентная запись	Нет

Это свойство используется для определения стиля вашего шрифта. Если существует уже готовый вариант шрифта (например, New York Oblique), то он будет применен, в противном случае изменение шрифта будет произведено программно.

Свойство font-va ria nt

Свойство	fontVariant
Значения	normal, small-caps
Значение по умолчанию	normal
Поддерживается	Всеми элементами
Работает ли механизм наследования	Да
Возможна ли процентная запись	Нет

Normal — это стандартный вид, установленный по умолчанию; small-caps использует прописные буквы того же размера, что и обычные строчные.

Свойство font-weight

Свойство	**fontWeight**
Значения	normal, bold, bolder, lighter или числовые значения от 100 до 900
Значение по умолчанию	normal
Поддерживается	Всеми элементами
Работает ли механизм наследования	Да
Возможна ли процентная запись	Нет

Определяет жирность шрифта текста, которая зависит от толщины линии. Если используется числовое выражение, то число обязательно должно быть кратным 100: 400 — то же, что и normal, 700 — то же, что и bold.

Свойства цвета и фона

Свойство color

Свойство	color
Значение	Название цвета или его номер
Значение по умолчанию	Зависит от браузера
Поддерживается	Всеми элементами

Работает ли механизм наследования	Да
Возможна ли процентная запись	Нет

Устанавливает цвет текста элемента страницы. Цвет может быть задан названием (например, green) или номером в соответствии со шкалой RGB. Это значение может быть задано несколькими способами: в шестнадцатеричной системе, например **"tfFFFFF",** процентами - "80%,20%,0%"; в десятеричной системе — "245,0,20".

Свойство background

Свойство	background
Значения	transparent, <цвет>, <11(?1_-адрес>, <repeat>, <прокрутка>, <положение>
Значение по умолчанию	transparent
Поддерживается	Всеми элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Да (относительно размера самого элемента)

Определяет параметры фона документа, transparent означает отсутствие фона. Вы можете использовать для фона либо цвет, либо изображение с заданным URL-адресом. Адрес может быть абсолютным или относительным, но его следует обязательно заключить в скобки и расположить непосредственно за словом **url:**

```
BODY { background: url (http://raumu.com/image/picture.gif) }
```

Можно использовать и цвет фона, и изображение. В таком случае рисунок будет расположен поверх цветного фона. Цвет может быть либо чистым, либо смешанным из двух. Изображение имеет несколько настроек:

- <repeat> может иметьзначения repeat, repeat-x, repeat-y и no-repeat; если эти значения не указаны, значением по умолчанию является repeat;
- <прокрутка> определяет, будет ли изображение оставаться на месте при прокручивании страницы или прокручиваться вместе с ней; возможные значения fixed, scroll;
- <положение> определяет расположение изображения на странице; значения могут быть процентными (горизонтальное, вертикальное), абсолютным расстоянием (горизонтальное, вертикальное) или определенными с помощью ключевых слов (допустимые значения: top, middle, bottom, left, center, right).

В следующих разделах описаны свойства, позволяющие задать свойства фона по отдельности.

Свойство background-attachment

Свойство	backgroundAttachment
Значения	fixed, scroll
Значение по умолчанию	scroll
Поддерживается	Всеми элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Нет

Определяет, будет ли прокручиваться фоновое изображение при прокрутке страницы или нет.

Свойство background-color

Свойство	backgroundColor
Значения	transparent, **<цвет>**
Значение по умолчанию	transparent
Поддерживается	Всеми элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Нет

Определяет цвет фона. Может быть один цвет или два смешанных цвета. Цвет задается названием (например, **green)** или номером по шкале RGB. Это значение может быть задано несколькими способами: в шестнадцатеричной системе (например, **"ttFFFFFF")** процентами — "80%,20%,0%"; в десятеричной системе — "255$_{д}$O". Синтаксис для использования двух смешанных цветов такой:

```
BODY { background-color: red/blue }
```

Свойство background-image

Свойство	backgroundImage
Значения	<URL-aflpec>, none
Значение по умолчанию	none
Поддерживается	Всеми элементами
Работаетли механизм наследования	Нет
Возможна ли процентная запись	Нет

С помощью этого свойства можно определить URL-адрес фонового изображения. Адрес может быть абсолютным или относительным, но его следует обязательно заключить в скобки и расположить непосредственно за словом **url.**

Свойства background-position

Свойства	backgroundPosition, backgroundPositionX, backgroundPositionY
Значения	**<положение>, <длина>,** top, **center,** bottom, left, right
Значение по умолчанию	top, left
Поддерживается	Всеми элементами
Работаетли механизм наследования	Нет
Возможна ли процентная запись	Нет

Указывают начальное положение фонового изображения с помощью двух значений, определяющих занимаемую часть страницы (по **горизонтали,** по вертикали), и абсолютное расстояние (в соответствующих единицах измерения, сначала по горизонтали, потом по вертикали), или используя два ключевых слова из возможных.

Свойство background-repeat

Свойство	backgroundRepeat
Значения	repeat, repeat-x, repeat-y, no-repeat
Значение по умолчанию	Repeat
Поддерживается	Всеми элементами

Работает ли механизм наследования	Нет
Возможна ли процентная запись	Нет

Определяет, повторяется ли фоновое изображение для заполнения страницы или элемента. Если используются значения repeat-x, repeat-y, то изображение повторяется лишь вдоль одного направления. По умолчанию устанавливается повторение по обоим направлениям.

Свойства списков

Свойство list-style

Свойство	oHstS tyle
Значения	<ключевое слово, <положение>, <URL-aflpec>
Значение по умолчанию	В зависимости от браузера
Поддерживается	Всеми элементами
Работает ли механизм наследования	Да
Возможна ли процентная запись	Нет

Определяет, как должны изображаться элементы списка. Может быть использовано для установки всех свойств. Их можно также установить по частям, пользуясь следующими свойствами.

Свойство list-style-image

Свойство	listStyleImage
Значения	none, <URL-aflpec>
Значение по умолчанию	попе
Поддерживается	Всеми элементами
Работает ли механизм наследования	Да
Возможна ли процентная запись	Нет

Определяет адрес изображения, используемого в качестве маркера или значка для каждого элемента списка.

Свойство list-style-position

Свойство	oИ stStyle Position
Значения	inside, outside
Значение по умолчанию	outside
Поддерживается	Всеми элементами
Работает ли механизм наследования	Да
Возможна ли процентная запись	Нет

Указывает, внутри или вне тела списка должен находиться маркер.

Свойство list-style-type

Свойство	listS tyleType
Значения	none, circle, disk, square, decimal, lower-alpha, upper-alpha, lower-roman, upper-roman
Значение по умолчанию	disk

Поддерживается	Всеми элементами
Работает ли механизм наследования	Да
Возможна ли процентная запись	Нет

Определяет вид маркера списка, используемого перед каждым элементом.

Свойства текста

Описания свойств, определяющих различные параметры отображения текста.

Интервал между буквами: свойство letter-spacing

Свойство	letterSpacing
Значения	normal, <длина>
Значение по умолчанию	normal
Поддерживается	Всеми элементами
Работает ли механизм наследования	Да
Возможна ли процентная запись	Нет

Определяет расстояние между буквами. Длина обозначает добавочное расстояние между буквами. Оно должно быть выражено с указанием единиц измерения.

Высота строки: свойство line-height

Свойство	lineHeight
Значения	<число>, <длина>, <проценты>, normal
Значение по умолчанию	В зависимости от браузера
Поддерживается	Всеми элементами
Работает ли механизм наследования	Да
Возможна ли процентная запись	Да, по отношению к размеру шрифта текущего элемента

Определяет высоту текущей строки. Числовые значения интерпретируются как размер шрифта текущего элемента, умноженный на соответствующий коэффициент (например, 1,2). Если используется длина, должны указываться единицы измерения. Процентное соотношение используется в сравнении с текущим размером шрифта и должно быть больше 100%.

Выравнивание текста: свойство text-align

Свойство	textAlign
Значения	left, right, center, justify
Значение по умолчанию	В зависимости от браузера
Поддерживается	Всеми элементами
Работает ли механизм наследования	Да
Возможна ли процентная запись	Нет

Определяет, как текст будет выровнен относительно содержащего его элемента. По существу, делает то же, что и тег **<DIVALIGN= .. .>**.

Украшение текста: свойства text-decoration

Свойства	textDecoration, textDecorationLineThrough, textDecorationUnderline, textDecorationOverline

Значения	none, overLine, underline, line-through
Значение по умолчанию	попе
Поддерживается	Всеми элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Нет

Позволяет добиться специального вида текста. Открыто для расширений, неопознанные расширения интерпретируются как подчеркивание. Это свойство не наследуется, но обычно распространяется на дочерние элементы, лежащие внутри родительских.

Красная строка: свойство text-indent

Свойство	textIndent
Значения	<длина>, <проценты>
Значение по умолчанию	zero
Поддерживается	Всеми элементами
Работает ли механизм наследования	Да
Возможна ли процентная запись	Да, относительно ширины родительского элемента

Устанавливает величину отступа в единицах измерения или в процентах от ширины родительского элемента.

Преобразование текста: свойство text-transform

Свойство	textTransform
Значения	capitalize, uppercase, lowercase, none
Значение по умолчанию	none
Поддерживается	Всеми элементами
Работает ли механизм наследования	Да
Возможна ли процентная запись	Нет

- capitalize делает заглавной первую букву каждого слова в текстовом элементе;
- uppercase преобразует все буквы в словах текстового элемента в заглавные;
- lowercase преобразует все буквы в словах текстового элемента в строчные;
- попе снимает все установки, приобретенные в результате наследования.

Расположение по вертикали: свойство vertical-align

Свойство	**verticalAlign**
Значения	baseline, sub, super, top, text-top, middle, bottom, text-bottom, <проценты>
Значение по умолчанию	baseline
Поддерживается	Встроенными элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Да, по отношению к высоте строки

Определяет расположение текущего текстового элемента по вертикали:

- baseline устанавливает выравнивание по образцу родительского элемента;

- middle устанавливает вертикальную среднюю линию текстового элемента на основе родительского элемента плюс половина строки последнего;
- super переводит текстовый элемент в верхний регистр;
- sub переводит текстовый элемент в нижний регистр;
- text-top выравнивает текстовый элемент по верху текста, набранного шрифтом родительского элемента;
- text-bottom выравнивает текстовый элемент по низу текста, набранного шрифтом родительского элемента;
- top выравнивает верх текстового элемента по верху самого высокого элемента текущей строки;
- bottom выравнивает низ текстового элемента по низу самого низкого элемента текущей строки.

Блочные свойства

Значения этих свойств используются для установки характеристик области вокруг элемента. Это может быть применено к символам, изображениям и т. д.

Свойства border-top-color, border-right-color, border-left-color, border-bottom-color, border-color

Свойства	borderTopColor, borderRightCoLor, borderLeftCobr, borderBottomColor, borderCoLor
Значение	<цвет>
Значение по умолчанию	<нет цвета>
Поддерживается	Блоками элементов и замещаемыми элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Нет

Определяют цвет четырех сторон рамок. Если вместо цвета подставить URL-адрес изображения, он будет повторяться, образуя рамку.

Свойства border-top-style, border-right-style, border-left-style, border-bottom-style, border-style

Свойства	borderTopStyle, borderRightStyle, borderLeftStyle, borderBottomStyle, borderStyle
Значения	none, solid, double, groove, ridge, inset, outset
Значение по умолчанию	none
Поддерживается	Блоками элементов и замещаемыми элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Нет

Определяют стиль четырех сторон рамок.

Свойства border-top, border-rignt, border-left, border-bottom, border

Свойства	borderTop, borderRignt, borderLeft, borderBottom, border
Значения	<ширина_рамки>, <стиль_рамки>, <цвет>
Значение по умолчанию	medium, none, <нет цвета>
Поддерживается	Блоками элементов и замещаемыми элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Нет

Определяют свойства четырех сторон рамок. Работают так же, как свойства отступов (см. ниже), за исключением того, что рамка видна:

- **<ширина_рамки>** может быть значением medium, thin или thick или задана в единицах измерения;
- <стиль_рамки> может иметь значение попе или solid.

Аргумент color используется для определения цвета заполнения фона элемента, пока он загружается, а также позади прозрачных частей элемента. Если передать вместо него адрес изображения, то рисунок будет повторяться, заполняя контур рамки.

Свойства border-top-width, border-right-width, border-left-width, border-bottom-width, border-width

Свойства	borderTopWidth, borderRightWidth, borderLeftWidth, borderBottomWidth, borderWidth
Значения	thin, medium, thick, <длина>
Значение по умолчанию	medium
Поддерживается	Блоками элементов и замещаемыми элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Нет

Определяют ширину рамки вокруг элемента. Каждая сторона может быть задана соответствующим свойством. Можно также заменить установку четырех индивидуальных свойств установкой одного свойства border-width так же, как и для свойств отступа margin.

Свойство clear

Свойство	clear
Значения	none, both, left, right
Значение по умолчанию	попе
Поддерживается	Всеми элементами
Работает ли механизм наследования	Нет
Возможнали процентная запись	Нет

Указывает, что следующие элементы должны быть показаны ниже элемента, выровненного по левому или правому краю. По умолчанию текст обтекает такие элементы.

Свойство clip

Свойство	clip
Значения	rect(<top><right><bottom><Left>), auto
Значение по умолчанию	auto
Поддерживается	Всеми элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Нет

Определяет, какая часть элемента видна. Все, что находится за пределами области, указанной этим свойством, увидеть нельзя.

Свойство display

Свойство	display
Значения	"", попе
Значение по умолчанию	«и
Поддерживается	Всеми элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Нет

Это свойство указывает, будет ли показан элемент.

Свойство float

Свойство	style Float
Значения	none, left, right
Значение по умолчанию	none
Поддерживается	Тегами **DIV,** SPAN и замененными элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Нет

Определяет обтекание элемента другими элементами слева или справа вместо помещения их под ним.

Свойства height

Свойства	height, pixelHeight, posHeight
Значения	auto, <длина>
Значение по умолчанию	auto
Поддерживается	Тегами DIV, SPAN и замененными элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Нет

Устанавливают высоту элемента и измеряют ее, если это необходимо. Значение возвращается в виде строки, включающей единицы измерения (px, % и т. д.). Для получения числового значения используется свойство posHeight.

Свойства left

Свойства	left, pixelLeft, posLeft
Значения	auto, <длина>, <проценты>

Значение по умолчанию	auto
Поддерживается	Всеми элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Да, относительно ширины родительского элемента

Задают горизонтальную координату элемента, позволяя корректно устанавливать и передвигать элементы. Значение возвращается как строка, включающая единицы измерения (px, % и т. д.). Для получения значения в виде числа используется свойство **posLeft.**

Свойства margin-top, margin-right, margin-left, margin-bottom, margin

Свойства	marginTop, marginRight, marginLeft, marginBottom, margin
Значения	auto, <длина>, <проценты>
Значение по умолчанию	zero •
Поддерживается	Блоками элементов и замещаемыми элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Да, относительно ширины родительского элемента

Определяют размеры отступов вокруг данного элемента. Свойство margin служит для задания четырех отступов. Если используется несколько значений, но меньше четырех, оставшиеся противоположные стороны будут равными. Эти значения устанавливают минимальное расстояние между текущим элементом и остальными.

Свойство overflow

Свойство	overflow
Значения	auto, visible, hidden, scroll
Значение по умолчанию	auto

Это свойство управляет тем, что произойдет, если содержимое элемента выйдет за его границы:

- visible — если содержимое и выйдет за пределы отведенного ему места, он все равно будет показан полностью; это значение принято по умолчанию;
- hidden — выступающие за границы элемента-контейнера части содержимого будут обрезаны;
- **auto** — добавляет полосы прокрутки, если содержимое выходит за границы элемента-контейнера;
- scroll — добавляет полосы прокрутки в любом случае.

Свойства padding-top, padding-right, padding-left, padding-bottom, padding

Свойства	paddingTop, paddingRight, paddingLeft, paddingBottom, padding
Значения	auto, <длина>, <проценты>

Значение по умолчанию	zero
Поддерживается	Блоками элементов и замещаемыми элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Да, относительно ширины родительского элемента

Определяют расстояние между содержимым и рамкой элемента. Свойство padding используется для задания четырех таких расстояний. Если имеется несколько значений, но меньше четырех, оставшиеся противоположные расстояния будут равными.

Свойство position

Свойство	position
Значения	absolute, relative, static
Значение по умолчанию	relative
Поддерживается	Всеми элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Нет

Устанавливает, каким образом вычисляется положение элемента в плоскости экрана:

- absolute — элемент будет поставлен в некоторое положение относительно фона и будет двигаться вместе с ним;
- static — аналогично относительному (relative), за исключением того, что при данном способе позиционирования элемент нельзя сместить с помощью параметров left и top;
- relative — элемент будет помещен в обычное положение относительно других в соответствии с положением в исходном коде (это значение установлено по умолчанию).

Свойства top

Свойства	top, pixelTop, posTop
Значения	auto, **<длина>, <проценты>**
Значение по умолчанию	auto
Поддерживается	Всеми элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Да, относительно ширины родительского элемента

Устанавливают или возвращают вертикальную координату элемента, позволяя корректно устанавливать и передвигать элементы. Значение возвращается как строка, включающая единицы измерения (px, % и т. д.). Для получения значения в виде числа используется свойство posTop.

Свойство visibility

Свойство	visibility
Значения	visible, hidden, inherit
Значение поумолчанию	inherit

Поддерживается	Всеми элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Нет

Позволяет элементу быть видимым или невидимым на странице. Невидимые (hidden) элементы занимают то же место и так же влияют на расположение других элементов, как и видимые (visible), но становятся прозрачными. Это свойство может быть использовано для отображения лишь одного из элементов, занимающих одно и то же место:

- visible — элемент виден на экране;
- hidden — элемент не виден на экране;
- inherit — элемент виден на экране, если его родительский элемент является видимым.

Свойства width

Свойства	width, pixelWidth, posWidth
Значения	auto, <длина>, <проценты>
Значение по умолчанию	auto за исключением элементов с внутренними установками размера
Поддерживается	DIV, SPAN и замещаемыми элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Да, относительно ширины родительского элемента

Устанавливают ширину элемента и измеряют ее, если это необходимо. Значение возвращается как строка, включающая тип единиц измерения (px, % и т. д.). Для получения значения в виде числа используется свойство posWidth.

Свойство z-index

Свойство	. zIndex
Значение	<число>
Значение по умолчанию	В зависимости от контекста в исходном тексте HTML
Поддерживается	Всеми элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Нет

Свойство z-index указывает, в каком порядке элементы будут перекрывать друг друга. Элементы с более высоким z-индексом появятся поверх элементов с более низким z-индексом. При положительных значениях элемент ставится перед обычным текстом, при отрицательных — позади него. Таким образом, на страницах создается набор плоских слоев, для которых задается порядок перекрывания.

Основные свойства печати

Свойство page-break-after

Свойство	pageBreakAfter
Значения	<auto>, <always>, <left>, <right>

Значение по умолчанию	<auto>
Поддерживается	Всеми элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Нет

Управляет началом и концом страницы, разрывом страницы и тем, где на странице содержание продолжится, то есть слева или справа.

Свойство page-break-before

Свойство	pageBreakBefore
Значения	<auto>, <always>, <left>, <right>
Значение по умолчанию	<auto>
Поддерживается	Всеми элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Нет

Управляет началом и концом страницы, разрывом страницы и тем, где на странице содержание продолжится, то есть слева или справа.

Свойства фильтров

Все фильтры вызываются с помощью ключевого слова **filter.** Есть два вида фильтров: статические и динамические. Динамические фильтры разделены на две группы: управляющие переходом (переходные) и управляющие появлением объекта на экране. Фильтры первой группы предназначены для того, чтобы объект накладывался надругие объекты или уходил с соседних объектов. Фильтры второй группы заставляют объекты появляться или исчезать одним из 23 возможных способов. Все фильтры вызываются одинаково из каскадной таблицы стилей:

filter: имя_фильтра (параметр!,. параметр2, и т. д.)

Существует 14 типов статических фильтров и 2 типа динамических.

Visualfilters

Свойство	объект.style.<имя_фильтра>(параметр! и т. д.)
Значения	<имя фильтра>(параметр1, параметр? и т. д.)
Значение по умолчанию	Нет примененного фильтра
Поддерживается	Всеми элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Нет

Управляет внешним видом объектов при помощи набора встроенных фильтров. Ниже в таблице приведен список доступных фильтров с указанием их функций.

Фильтр	Функция
alpha	Управляет степенью видимости объекта
blur	Создает эффект движения объекта
chroma	Переводит определенный цвет изображения в прозрачный (transparent)
dropShadow	Отображает силуэт определенного цвета для выбранного объекта, создавая иллюзию, что объект находится над страницей, отбрасывая тень

Фильтр	Функция
fliph	Отражает объект симметрично центральной горизонтальной оси
flipv	Отражает объект симметрично центральной вертикальной оси
glow	Создает для объекта эффект сияния
grey	Отображает объект в градациях серого
invert	Обращает цвета объекта
Light	Моделирует освещение объекта различными источниками цвета
mask	Отображает прозрачные (transparent) символы определенным цветом, а непрозрачные делает прозрачными
redirect	Конвертирует объект в DAImage-объект, то есть в изображение, которым может манипулировать Microsoft DirectAnimation
shadow	Отрисовывает «боковые» грани объекта, создавая впечатление объемности
wave	Выполняет синусоидальное преобразование объекта вдоль вертикальной оси
xray	Изменяет глубину цвета объекта и отрисовывает черно-белым, делая его похожим на рентгеновский снимок

revealTrans

Свойство	объект,style. revealtrans(duration = <длительность>, transition=<типперехода>)
Значения	revealtrans (duration = <длительность>, transition =<тип перехода>)
Значение по умолчанию	Нет примененного фильтра
Поддерживается	Всеми элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Нет

Позволяет открывать или закрывать одни элементы другими, используя 23 встроенных шаблона (см. табл. ниже).

- <длительность> — время, за которое превращение с помощью фильтра произойдет полностью (в миллисекундах).
- <тип перехода> — целое число, соответствующее номеру применяемого фильтра.

Тип превращения	Номер фильтра
Box In (стягивающийся прямоугольник)	0
Box Out (расширяющийся прямоугольник)	1
Circle In (стягивающийся круг)	2
Circle Out (расширяющийся круг)	3
Wipe Up (стирание вверх)	4
Wipe Down (стирание вниз)	5
Wipe Right (стирание вправо)	6
Wipe Left (стирание влево)	7
Vertical Blinds (вертикальные жалюзи)	8

Продолжение ⇨

Тип превращения	Номер фильтра
Horisontal Blinds (горизонтальные жалюзи)	9
Checkerboard Across (сужающиеся клетки шахматной доски)	10
Checkerboard Down (закрывающаяся шахматная доска)	11
Random Dissolve (случайный наплыв)	12
Split Vertical In (вертикальное деление внутрь)	13
Split Vertical Out (вертикальное деление наружу)	14
Split Horisontal In (горизонтальное деление внутрь)	15
Split Horisontal Out (горизонтальное деление наружу)	16
Strips Left Down (стирание влево-вниз)	17
Strips Left Up (стирание влево-вверх)	18
Strips Right Down (стирание вправо-вниз)	19
Strips Right Up (стирание вправо-вверх)	20
Random Bars Horisontal (случайные горизонтальные полосы)	21
Random Bars Vertical (случайные вертикальные полосы)	22
Random selection of (0-22) (случайный выбор из предыдущих вариантов)	23

blendTrans

Свойство	объект. style. blendtrans
Значения	blendTrans(duration= <длительность>), (duration =<длительность>)
Значение по умолчанию	Нет примененного фильтра
Поддерживается	Всеми элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Нет

Позволяет управлять появлением и исчезновением выбранных видимых объектов. Здесь <длительность> — время, за которое действие фильтра произойдет полностью (в миллисекундах).

Свойство cursor

Свойство	cursor
Значения	auto, crosshair, default, hand, move, e-resize, ne-resize, nw-resize, n-resize, se-resize, sw-resize, s-resize, w-resize, text, wait, help
Значение по умолчанию	auto
Поддерживается	Всеми элементами
Работает ли механизм наследования	Нет
Возможна ли процентная запись	Нет

Определяет вид указателя мыши.

События динамического HTML

Событие	Описание
onabort	Пользователь прерывает загрузку изображения
onafterupdate	Окончание передачи данных
onBeforeunload	Происходит перед выгрузкой страницы, что позволяет сохранить данные
onbeforeupdate	Происходит перед обновлением страницы
onblur	Выход окна из фокуса
onbounce	Происходит в элементе <MARQUEE>, когда значение свойства BEHAVIOR — ALTERNATE, и содержимое доходит до конца
onchange	Изменение содержимого элемента
onclick	Щелчок на документе
ondataavauable	Происходит периодически, когда данные приходят из асинхронного источника
ondatasetchanged	Происходит при изменении порядка данных, например во время фильтрации данных
ondatasetcomplete	Происходит, когда все данные из источника становятся доступными
ondbLclick	Пользователь выполнил двойной щелчок на документе
ondragstart	Пользователь начинает перетаскивание
onerror	Ошибка во время передачи
onerrorupdate	Происходит при отмене изменения данных событием onbeforeupdate, заменяя событие onafterupdate
onfilterchange	Происходит при смене состояния статического фильтра или при окончании перехода динамического
onfilterevent	Завершен данный переход фильтра
onfinish	Происходит по окончании цикла в элементе <MARQUEE>
onfocus	Элемент становится активным
onhelp	Нажатие пользователем клавиши F1 или кнопки Help
onkeydown	При нажатии клавиши
onkeypress	Происходит при нажатии клавиши и продолжается при удержании
onkeyup	Пользователь отпускает клавишу
onload	Полная загрузка элемента
onmousedown	Пользователь нажимает на кнопку мыши
onmousemove	Происходит, когда пользователь двигает мышь
onmouseout	Указатель мыши выходит из области элемента
onmouseover	Указатель мыши впервые попадает в область на элемент
onmouseup	Пользователь отпускает кнопку мыши
onreadystatechange	Изменение свойства readystate
onreset	Очистка формы

продолжение ⇨

Событие	Описание
onresize	Изменение пользователем размеров окна
onrowenter	Происходит при изменении данных в текущей строке
onrowexit	Происходит перед изменением данных в текущей строке источником
onscroll	Прокручивание окна пользователем
onselect	Изменение текущего выделения
onselectstart	Первый запуск пользователем выделенной части документа
onstart	Прокрутка бегущей строки элемента MARQUEE
onsubmit	Отсылка формы на сервер или нажатии кнопки SUBMIT (Отправка)
onUnload	Происходит непосредственно перед выгрузкой страницы

Алфавитный указатель

N

Q

R

S

U

V

W

Z

А

Б

В

Г

Д

З

И

К

Л

М

Н

О

П

Р

С

Т

У

Ф

Ц

Ч

Ш

Э

Я

Вадим Дунаев

Самоучитель JavaScript

2-е издание

Главный редактор *Е. Строганова*
Заведующий редакцией *А. Кривцов*
Руководитель проекта *В. Шачин*
Художник *Н. Биржаков, В. Медведев*
Корректоры *С. Беляева, А. Моносов*
Верстка *Л. Харитонов*

Лицензия ИД № 05784 от 07.09.01.
ООО «Питер Принт». 194044, Санкт-Петербург, пр. Б. Сампсониевский, д. 29а.
Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2; 953005 — литература учебная.
Подписано в печать 20.01.05. Формат 70Х100/16. Усл. п. л. 32,25. Тираж 3000 экз. Заказ № 9227.
Отпечатано с готовых диапозитивов в ФГУП «Печатный двор» им. А. М. Горького
Министерства РФ по делам печати, телерадиовещания и средств массовых коммуникаций.
197110, Санкт-Петербург, Чкаловский пр., 15.



АНТИВИРУС ИГОРЯ ДАНИЛОВА



Вадим Дунаев

(САМОУЧИТЕЛЬ)

JavaScript

2-е издание

Эта книга поможет вам освоить.-

- общиепринципыпрограммирования;
- основные элементыязыка JavaScript;
- алгоритмы и методы веб-программирования.

Книга предназначена для самостоятельного освоения программирования на языке JavaScript. Она содержит множество примеров и текстов готовых к использованию программ. Рассматриваются вопросы создания сценариев для веб-сайтов, а также сценариев, выполняемых Windows Scripting Host. В приложениях приводится справочная информация по JavaScript и HTML Издание адресовано как новичкам, так и **тем, кто** уже имеет некоторый опыт в веб-дизайне и программировании.

Изучите один из популярнейшихязыков для веб-программирования самостоятельно!

ШППТЕР®

Заказ книг:

197198, Санкт-Петербург, а/я 619
тел.: (812)103-73-74, postbook@piter.com
61093, Харьков-93, а/я 9130
тел.: (057)712-27-05, piter@kharkov.piter.com

www.piter.com - - вся информация о книгах и веб-магазин

ISBN 5-469-00804-5

9 785469 008040

- способствуют быстрейшему профессиональному и
- духовному росту читателей и являются рекламой
- бумажных изданий таких документов.
-
- Все авторские права сохраняются за правообладателем.
- Если Вы являетесь автором данного документа и хотите
- дополнить его или изменить, уточнить реквизиты автора
- или опубликовать другие документы, пожалуйста,
- свяжитесь с нами по e-mail - мы будем рады услышать ваши
-

пожелания.